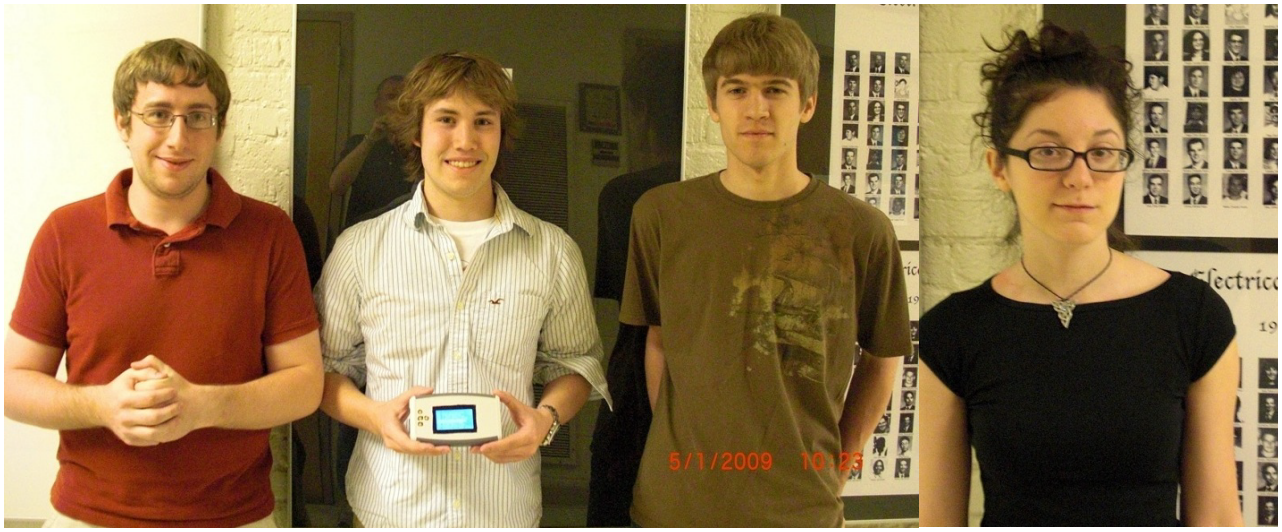
ECE 477 Final Report – Spring 2009
Team 9 – FLACtrac



Isaac Jones Greg McCoy Brett Mravec Danielle Miller

Team Members:

#1: Greg McCoy **Signature:** _____ **Date:** _____

#2: Brett Mravec **Signature:** _____ **Date:** _____

#3: Isaac Jones **Signature:** _____ **Date:** _____

#4: Danielle Miller **Signature:** _____ **Date:** _____

CRITERION	SCORE	MPY	PTS
Technical content	0 1 2 3 4 5 6 7 8 9 10	3	
Design documentation	0 1 2 3 4 5 6 7 8 9 10	3	
Technical writing style	0 1 2 3 4 5 6 7 8 9 10	2	
Contributions	0 1 2 3 4 5 6 7 8 9 10	1	
Editing	0 1 2 3 4 5 6 7 8 9 10	1	
Comments:		**TOTAL**	

TABLE OF CONTENTS

Abstract 1
1.0 Project Overview and Block Diagram 1
2.0 Team Success Criteria and Fulfillment 2
3.0 Constraint Analysis and Component Selection 3
4.0 Patent Liability Analysis 8
5.0 Reliability and Safety Analysis 11
6.0 Ethical and Environmental Impact Analysis 15
7.0 Packaging Design Considerations 18
8.0 Schematic Design Considerations 23
9.0 PCB Layout Design Considerations 26
10.0 Software Design Considerations 28
11.0 Version 2 Changes 31
12.0 Summary and Conclusions 32
13.0 References 33
Appendix A: Individual Contributions A-1
Appendix B: Packaging B-1
Appendix C: Schematic C-1
Appendix D: PCB Layout Top and Bottom Copper D-1
Appendix E: Parts List Spreadsheet E-1
Appendix F: Software Listing F-1
Appendix G: FMECA Worksheet G-1

Abstract

The FLACtrac is a mobile digital audio player which decodes FLAC (Free Lossless Audio Codec) files stored on a SecureDigital (SD) card and synthesizes the sound data to a 3.5mm headphone jack which could be routed to earphones or external speakers. To facilitate user control of the device, an LCD display and four pushbuttons form the human interface. The LCD display shows useful information such as a list of files on the memory card or the metadata for an audio file being played. FLAC is notable for providing an open-source compression mechanism for reducing the file size while still maintaining perfect integrity of the original audio data.

1.0 Project Overview and Block Diagram

The FLACtrac is built around the Analog Devices ADSP-21262 SHARC Digital Signal Processor. The DSP has no onboard, nonvolatile memory, so the software is loaded at boot from an Atmel AT25F2048N SPI Flash EEPROM. Audio data is loaded into the device via a removable SD (SecureDigital) card. The SHARC DSP interfaces directly to the SD card because they both support 3.3V SPI communication. The DSP then can perform the FLAC decoding and output the analog audio through an I^2S interface to the AD1854 D/A and SSM2135 amplifier to headphones or another analog destination. Visual feedback to the user is provided on a 128 by 64 pixel Crystalfontz monochrome LCD display. The DSP drives the LCD module's parallel interface by using SPI to load a 74HC595 shift register with the appropriate parallel data and then using general purpose input/output (GPIO) pins to clock the data transfers. The pushbuttons are connected to GPIO pins as well, and polled to determine if the user is pressing a button.

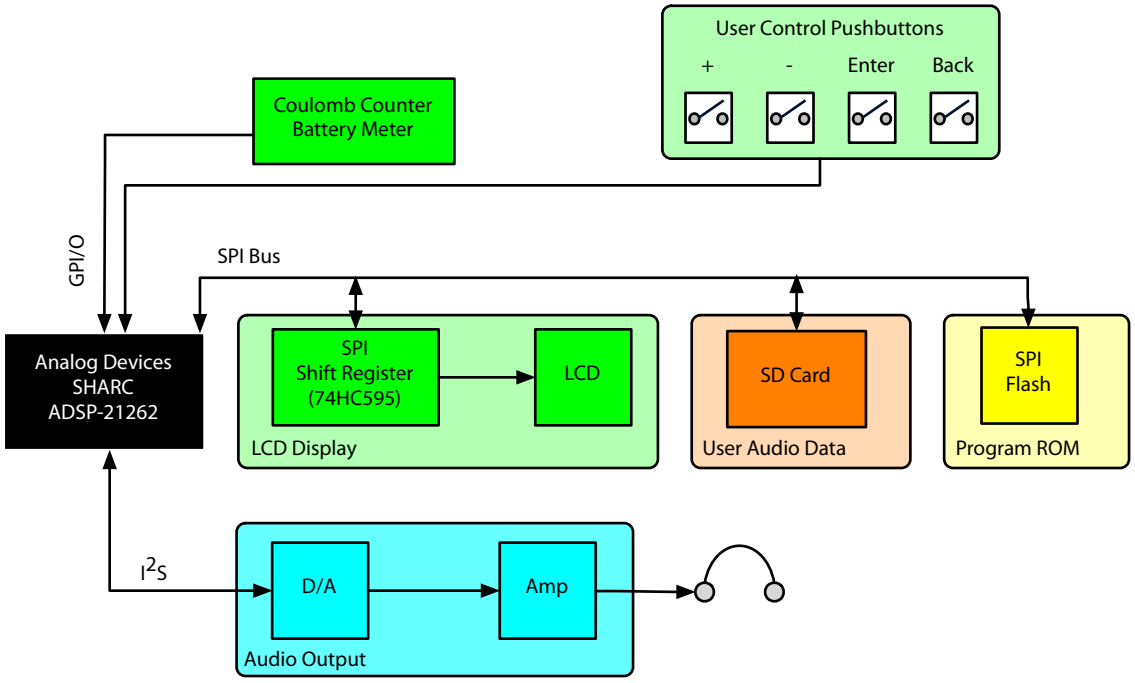


Figure 1. Block Diagram

2.0 Team Success Criteria and Fulfillment

Table 1 below lists our project-specific success criteria, and our results in meeting them.

PSSC	Status	Comments
An ability to decode files stored in the FLAC format	Partially Complete	Noise on the SPI bus prevents sufficient SD transfer throughput to decode FLAC at the speed of the original recording
An ability to select files stored on the device	Complete	SD card communications and FAT32 filesystem implemented, LCD shows file listing.
An ability to display visualization information	Complete	LCD shows “waveform” display of decoded audio, as well as file metadata.
An ability to output audio to a headphone or speaker port	Complete	Digital PCM audio can be converted by the D/A, LP filtered and appropriately amplified to a headphone jack.
An ability to pause and resume playback on user input.	Complete	User can press Play/Pause button to pause and resume playback.

Table 1. PSSC Completion Status

3.0 Constraint Analysis and Component Selection

3.1 Computation Requirements

The computational performance of our device is highly critical to its success, because it must be capable of decoding the stored FLAC data in real-time. To help determine the computational requirements of our FLAC decoding algorithm, we developed a prototype implementation in C by adapting existing from an open-source reference implementation. From testing this implementation we have gathered performance data that indicates the computational resources required to successfully decode and play FLAC files. In addition, we need to generate visualization data for output to the LCD in real-time.

3.2 Interface Requirements

To be useful, the proud owner of a FLACtrac must be able to load their FLAC files onto the device. Initially, we envisioned providing a USB port for the user to attach a mass storage device with FLAC files loaded on it, but the clumsiness of having a USB flash drive sticking out of a portable device seemed inappropriate. Instead, loading audio data is accomplished with a SecureDigital (SD) card slot. All SD cards can be accessed using either a unique SD communications protocol or the more ubiquitous SPI protocol, which sacrifices some bandwidth in favor of compatibility [1]. Since the proprietary SD interface for high-speed bulk data transfers is significantly more complicated to implement, and the SPI mode is sufficiently fast for real-time playing, we plan to use the SPI interface mode. This also enables the user to carry an arbitrarily large number of inexpensive SD cards rather than relying on a fixed internal memory.

The decoded FLAC data from the SD card then needs to be synthesized to analog form for connection to headphones or speakers. Therefore, a digital-to-analog converter is necessary (DAC). Some DSPs we examined accomplish this by using fast on-board pulse-width modulation (PWM) into a switching "Class-D" amplifier, which is then output through a low-pass filter. Others provide a digital audio interface (such as I^2S), which facilitates connecting peripheral D/A converters that employ other techniques, such as Delta-Sigma conversion.

To display information to the user, we need to use a liquid-crystal display (LCD). There are several varieties of these available, some serial (which tend to have partitioned cells for displaying individual characters) and some parallel (which are more suited towards graphics). We want to have a more flexible graphical LCD to help facilitate displaying visualization information, which is one of our PSSCs. Since the graphical LCD modules we have found almost exclusively have parallel interfaces, we will need to supply an eight-bit parallel port along with a few (no more than five) control lines to facilitate this.

We also need to get information from the user, who will operate pushbuttons to facilitate file selection, volume control, etc. There must be enough general-purpose digital I/O to connect these components or we will need additional circuitry to serialize the user input.

3.3 On-Chip Peripheral Requirements

Based on the requirements enumerated in Section 3.2, we will need an SPI master interface which can enable at least two different devices, an eight-bit parallel interface (with no more than five separate control signals), and either two PWMs fast enough to reproduce the audio, or a dedicated digital audio interface (such as I^2S or a dedicated SPI) to connect to a D/A converter.

3.4 Off-Chip Peripheral Requirements

"Glue" circuitry is needed between the analog audio output and the user input devices. For the analog audio output, if a fast PWM output were used, circuitry to filter and amplify the signal would be necessary. If a digital audio interface is used, a matching D/A converter must be supplied in addition. For the user control surfaces (buttons, knobs, etc.) an off-board serial encoding device would be needed if there are not enough general-purpose I/O pins available. Similarly, if there are not enough parallel ports available, we may need to use a shift register to convert a serial output pin from the DSP to a parallel interface on an LCD. The SD card is able to communicate directly using SPI without any glue logic if the DSP supports SPI at the 3.3V required by the SD standard.

3.5 Power Constraints

As a mobile device, an internal rechargeable battery will power the FLACtrac. Therefore, keeping power consumption of all components is an important consideration.

3.6 Packaging Constraints

Again, the FLACtrac is intended to be a portable device, so size and weight are important constraints. The Apple iPod nano is a similar device in terms of functionality and purpose, and is probably the smallest (or smaller than the) desirable size. It measures 3.6" tall, 1.5" wide, and 0.24" deep (90.7 x 38.7 x 6.2 mm); and weighs 1.3 ounces (36.8 grams) Our general impression is that the device could weigh several ounces, but more than 8-10 ounces or so and the device would become cumbersome to carry. [2].

3.7 Cost Constraints

The digital audio player market is highly saturated, so cost is a major constraint on the success of a product. Again, the benchmark device in the sector is the iPod. The iPod nano is the most similar device to the planned design of the FLACtrac and is widely available for an MSRP of $149 for the 8 GB model. Other digital audio players are roughly the same range. We will attempt to meet this cost target [2].

3.8 Component Selection Rationale

We looked at a few different DSP families in depth, in particular the ADSP-21xx [3] series and the SHARC family from Analog Devices [4], and the Freescale DSP563xx series [5]. We chose to evaluate these families because they are designed for audio applications and provide peripherals and libraries designed to ease development. Table 2 shows the differences between the devices we evaluated:

	Analog Devices ADSP-21991	Analog Devices ADSP-21262 (SHARC)	Freescale DSP56362
Clock	160 MHz	200 MHz	100 MHz
Program Memory	512kb	4Mb	720kb
Data Memory	256kb	2Mb	120kb
Digital I/O	16-Bit General Purpose I/O, Synchronous Serial and SPI interface with up to 7 slaves	16-bit Parallel Port, SPI interface with up to 4 slaves	8-bit Parallel Port, SPI interface (1 slave)
Audio Capability	On-board A/D and PWM-based D/A	On-board high speed Digital Audio Interface (with I^2S)	On-board "Enhanced Serial Audio Interface" (with I^2S), Digital Audio Transmitter (with S/PDIF)
Core Voltage	2.5 V	1.2 V	3.3 V
Package	176-LQFP	144-LQFP	144-LQFP
Development Tools/Evaluation Board	Not Available, very new product	Available & Acquired	Available for similar processors in family
Cost	$26.36	$18.01	$7.40

Table 2. Digital Signal Processor Comparison

We ultimately chose the Analog Devices ADSP-21262 processor, which we will refer to from now on as the "SHARC". It has far more memory and computational capability than the other processors. The other two processors also have a lot of extraneous features that we don't need, such as A/D conversion, and S/PDIF interfaces. On the other hand, the SHARC DSP doesn't have onboard D/A conversion, so an external D/A converter that can connect to the I^2S interface will need to be supplied. Another big factor in selecting the

SHARC is that development boards are already locally available so we can start designing and evaluating. The ADSP-21991 is a very new DSP and a development board is not yet available for it. The A development board is available for family members of the Freescale DSP, but we don't have it on hand. The SHARC processor meets all of our requirements, the only compromise being that the ADSP-21991 is more appropriate for low-power designs.

Unfortunately, the SHARC only has one 16-bit parallel port, which can be configured as general-purpose I/O pins. We need to be able to connect both our parallel LCD and our user controls. To overcome this conflict we plan to connect the LCD using a shift register (either standard logic or a programmed PLD) to the SPI interface, forming a serial-to-parallel converter. This will free up the native parallel port on the SHARC for use as general purpose I/O connected to our switches and other controls.

Analog Devices makes a large selection of D/A converters designed for audio applications, which integrate nicely with the SHARC family through their common digital audio interface. For our application, we chose to use the AD1854, which is a complete single chip stereo audio sigma-delta D/A. Coupling this with the matching SSM2135 audio operational amplifier will form our audio output stage.

3.9 Summary

We chose to use the Analog Devices ADSP-21262 SHARC digital signal processor primarily because it best meets our computational and peripheral requirements for our application compared to other DSPs we are aware of, and also has good local development support. The bulk of our digital I/O will take place on the SPI interface, which will carry the SD memory card and the shift register feeding the graphical LCD display. The D/A conversion will connect to the dedicated digital audio interface on the SHARC. The user controls (pushbuttons, etc.) will be connected to the parallel port on the SHARC. The cost of the components specified so far is about $60, which should put us close to being on target for meeting our price point of $150, especially considering that our prices reflect the cost of single orders of several components. For example, the LCD display decreases

from $27 for one unit to around $7 for orders of 100 or more, so in a mass-produced setting our current cost situation would be even more desirable.

The final parts list/bill of materials is located in Appendix E.

4.0 Patent Liability Analysis

4.1 Results of Patent and Product Search

As mentioned above, our device has notable similarities to current production devices such as the Apple iPod and Sony MiniDisc player. A primary function of both of these devices is to play audio back to the user. Since it is possible that software algorithms will determine the patent infringement, it is important to note which encoding algorithms each device supports. The iPod is configured to use MP3, AAC/MP4, Protected AAC, AIFF, WAV, Audible audiobook, and Apple Lossless codecs. The MiniDisc player uses ATRAC, ATRAC3, ATRAC3plus, and MP3 codecs. It is also important to consider the hardware used in the player to determine patent liability. Both the MiniDisc player and the iPod use hardware based decoding ICs. In addition to this, it is important to consider that the MiniDisc player reads data from a magneto-optical disc.

In addition to the prior products that display similar functionality to our device, there are a fair number of patents that cover functions performed by our device. The relevant patents are: US05392265, US06515212, and US06252947.

US05392265, filed May 17, 1993, covers Recording and Reproducing Apparatus which Calculates and Displays Management Information of Recorded Segments. The above apparatus is one on which musical information is recorded discretely together with management information for managing the information. The apparatus calculates, based on the management information, a playing time of each musical piece, a total playing time of all the musical pieces, a remaining time for which the recording medium can be further recorded, and a remaining number of pieces which can be recorded and displays the result of said calculations. Further inspection of the claim language used in the patent specifically mentions performing these calculations based on the management information concerning the recording medium, not the management information of the files

themselves. The claim that has the greatest potential for infringement is claim number 9[1].

US06515212, filed February 4, 2003, covers an information medium configured to transcode data stored in various encoding formats and send the decoded data to an audio output unit either digital or analog. The claim that has the greatest potential for infringement is claim number 9[2].

US06252947, filed June 26, 2001, covers a system and method for playing back data segments managed by one or more playback servers. The system and method are also capable of playing back the segments in a specified order and such that gaps between the segments are minimized. The claim that holds the most potential for infringement is claim number 2[3].

4.2 Analysis of Patent Liability

Though our device has significant potential for patent infringement, further analysis shows that we do not, in fact, infringe on any patents. Due to major design differences, we avoid both literal infringement and doctrine of equivalents infringements on both the devices we are similar to and the patents whose functions we are similar to.

Though Sony's MiniDisc player seems, at first glance to be very similar to our device, further analysis proves significant dissimilarity. The best way to approach this is to acknowledge that we perform substantially the same function, which we do, of playing audio stored on a removable storage device in a mobile device. However, we do not perform this function in a substantially similar fashion. Firstly, we do not use magneto-optical disks. Our design calls for removable, but solid state memory (SD cards). This is a substantially different function since our design is not subject to shock errors, and thus does not need the built-in bUFfering that the MiniDisc player requires to ensure uninterrupted playback. In addition to this, our device uses a software algorithm to decode FLAC files stored on the device, as opposed to the MiniDisc device, which uses a hard-ware based chip for decoding. Between the read method and the decode method, our device performs substantially the same function, but in a substantially different way and thus does not violate the doctrine of equivalents for the MiniDisc player.

A similar analysis can be applied to the iPod device from Apple. Since the iPod does not have removable media, the type of storage media is irrelevant. However, both players (ours and the iPod) decode audio data. In a similar to fashion to the MiniDisc player, the iPod uses a hardware chip to decode the audio data stored on its internal (non-removable) drive and, again, our software algorithm is substantially different. Thus, our device also passes the doctrine of equivalents with the iPod device.

The patents that ours device possible infringes upon can be similarly analyzed. However, this analysis is much easier as the patents clearly state what is claimed under the patent. For Patent US05392265, claim number 9 is the one that has the highest risk of violation. However, upon further analysis of the claim it is obvious that our design does not actually violate said claim, since the claim specifically denotes computing time metadata from information stored in the management table of the storage medium. In our design, we calculate time metadata based on information stored in the file header of the FLAC file we are currently playing. Thus, we perform substantially the same function in a substantially different fashion and do not violate either a literal claim or a doctrine of equivalents claim.

In a similar fashion, claim 9 of patent 6515212 can be proven to be sufficiently different. In the text of claim 9 and in fact the entire patent, non-compressed data processed in parallel with compressed data in the case that the compression algorithm is not supported is specifically mentioned. Since we do not use non-compressed data in parallel, our device does not violate. However, in the case where we do violate, the patent would cover every music-playing device on the market, and would thus not be specific enough to actually prosecute any infringement claims.

Lastly, claim 2 of patent number 6252947 can be proven to be sufficiently different. This particular patent is a software patent in addition to a hardware patent. Specifically, this patent covers a method of communication between server and client devices to formulate uninterrupted playback of a file on the client device. Since our device does not communicate with a server in any way, shape, or form, we do not violate this claim of this patent. Again, the only way to claim that our device violates this particular patent would be to assert that the SD card counts as a server for the device. In this case, any device that

communicates with any type of memory would be considered to violate this patent, and would thus be unenforceable.

4.3 Action Recommended

Though no infringement exists, it is possible that, with a very liberal interpretation of the last two patents discussed, we violate these patents. If this were the case, it would be very unlikely the patents would actually be enforceable since it is so general and covers such a wide range of devices.

4.4 Summary

In summary, though the FLACtrac has potential for patent violation, it does not violate any existing patents due to its major design differences. The greatest boon to the FLACtrac's operation is that its fundamental operating component, the FLAC codec, is open source, and thus not covered by any patents. In addition, the decision to implement the device in a DSP, and thus software, rather than in hardware also assisted in mitigating patent liability.

5.0 Reliability and Safety Analysis

5.1 Reliability Analysis

The FLACtrac poses very little threat to the welfare of its user and only has the possibility for minimal injury to occur. Reliability is not a big issue either as in today's world music player devices are a kind of throw away device. The average user buys one then in a few years gets an upgrade to a newer, smaller and larger capacity model. This allows us to be more lax on the reliability requirements and pass the savings of less complex circuits on to the consumer. The reliability of a product can be split up into the reliability of its more complex parts. One can then use these parts to approximate the reliability of the whole. This section analyzes four components that are either complex or operate at above ambient temperatures. These are the components that are most likely to fail and are used to closely approximate the failure rates of the entire device.

The first component is the Analog Devices ADSP-21262 SHARC DSP. It is the most complex part on our board and contains 144 pins. This makes it a prime target for this type of analysis.

Parameter Name	Description	Value	Comments
C_1	Die Complexity Failure Rate	.56	32 -bit microcontroller
C_2	Package Failure Rate	0.077	144 pin SMT $C_2 = 3.6*10^{-4} * (Np)^{1.08}$
π_T	Temperature Factor	3.1	T_J = 125 °C for microcontroller
π_E	Environmental Factor	0.5	Ground Benign Environment
π_Q	Quality Factor	10	Commercial Product
π_L	Learning Factor	1	In production for ≥ 2 years
λ_p	Failures per 10^6 hours	17.745	
MTTF	Mean time until failure	56353 Hours or 6.4 years	

Table 3. Analog Devices ADSP-21262 SHARC DSP Reliability Analysis

The next two parts are two of the power regulators: the Texas Instruments TPS63030 3.3V Regulator and the Linear Technology LT1302 5V Boost regulator. These parts are most likely to get hot and thus have higher failure rates.

Parameter Name	Description	Value	Comments
C_1	Die Complexity Failure Rate	0.020	101-300 Transistors
C_2	Package Failure Rate	0.0043	10 pin SMT
π_T	Temperature Factor	3.1	T_J = 125 °C
π_E	Environmental Factor	0.5	Ground Benign Environment
π_Q	Quality Factor	10	Commercial Product
π_L	Learning Factor	1	In production for ≥ 2 years
λ_p	Failures per 10^6 hours	0.6415	
MTTF	Mean time until failure	1558846 Hours or 177.9 years	

Table 4. Texas Instruments TPS63030 Reliability Analysis

Parameter Name	Description	Value	Comments
C_1	Die Complexity Failure Rate	0.020	101-300 Transistors
C_2	Package Failure Rate	0.0034	8 pin SMT
π_T	Temperature Factor	3.1	T_J = 125 °C
π_E	Environmental Factor	0.5	Ground Benign Environment
π_Q	Quality Factor	10	Commercial Product
π_L	Learning Factor	1	In production for ≥ 2 years
λ_p	Failures per 10^6 hours	0.637	
MTTF	Mean time until failure	1569858 Hours or 179.2 years	

Table 5. Linear Technology LT1302 Reliability Analysis

The last component is the Linear Technology LTC4054 battery charging IC. This device is chosen because it could get hot when the battery is being charged and it has one of the only components that could cause harm to the user, by overcharging and leading to catastrophic battery failure.

Parameter Name	Description	Value	Comments
C_1	Die Complexity Failure Rate	0.020	101-300 Transistors
C_2	Package Failure Rate	0.00205	5 pin SMT
π_T	Temperature Factor	3.1	T_J = 125 °C
π_E	Environmental Factor	0.5	Ground Benign Environment
π_Q	Quality Factor	10	Commercial Product
π_L	Learning Factor	1	In production for ≥ 2 years
λ_p	Failures per 10^6 hours	0.63025	
MTTF	Mean time until failure	1586671 Hours or 181.1 years	

Table 6. Linear Technology LTC4054 Reliability Analysis

After finding the mean time until failures, it is obvious that a few of the components do not fulfill the requirements for reliability. The main concern for improving reliability would be to prevent injury to the user of the device. Although the battery charging IC has a very low failure rate, it does not meet the $\lambda < 10\text{-}9$ requirement. It is not essential that the device not fail, only that the failure that occurs does not injure the user. This would involve some kind of extra control to insure that the battery could not be overcharged. If this could not happen, then the failure rate by the chip would be acceptable. The rest of the components would only require that they are not operating at the extremes given in the

data sheets. If this were true, the failure rates would be a lot lower and the products would be a lot more reliable. The DSP has the highest failure rate by far so that is the limiting factor in the field that would cause the most device failures. Some steps could be taken to insure that failures of this part of the system would not cause harm to the user. The only way that this could happen would be through very loud sound output. After addressing this issue, the only problem with a failure of the DSP would be a failure of the system.

5.2 Failure Mode, Effects, and Criticality Analysis (FMECA)

For analysis, the circuit is divided into functional blocks. For simplicity, we are using four blocks. Block A is the power components, Block B is the audio circuitry, Block C is the LCD, and block D is the DSP. These blocks make it easy to observe the different failure modes and effects they have on the operation of the system. The schematics of each of these blocks is shown in Appendix C.

The FLACtrac is a small consumer product which in normal use does not pose any threat to the user. This allows us to have a simplified view over criticality, where have a high criticality level that only could be caused by the failure of one of the analyzed components. This failure, however, could cause harm, so an acceptable error rate for this to occur is $\lambda < 10^{-9}$. The rest of the criticality levels are harmless to the user and thus have a much higher acceptable error rate. The lower two criticality levels are defined by how their failure affects the rest of the system. The medium criticality level is described as the level in which a failure effects the overall operation of the rest of the system. Because of this, the acceptable error rate is defined as $\lambda < 10^{-6}$. The lowest criticality level is defined as a failure that only causes loss of the functionality of the specific block where the failure occurred. For this the acceptable error rate is $\lambda < 10^{-5}$. With these levels defined in this way, it is possible to design a system that is safe to use and reliable to use and fix in normal operating conditions. The FMECA worksheets are included in Appendix G.

5.3 Summary

The FLACtrac would be a consumer device that is not likely to cause injury in the case of a failure of any component on the board. The two cases which could cause injury would be the overcharging of the battery and unintentionally loud audio playback. With some

additional precautions, the effects of these critical failures can be reduced to a failure rate of 'never' happening. After looking at some of the critical components of the system, it showed that the DSP was the chip that was most likely to fail. This makes it a limiting factor that makes the reliability of the product as a whole similar to that of the DSP itself. Although at its current state the rate of failure seems high, it is a very conservative estimate and is fairly reasonable for a consumer device like the one we are producing which inherently have a quick technology depreciation rate.

6.0 Ethical and Environmental Impact Analysis

6.1 Introduction

As sustainable development has become a dominant economic, environmental, and social issue of the 21st century [1], both the ethical and environmental impact of the FLACtrac must be analyzed. This entails evaluating not only design integrity, but possible risk of components to the user, the environment, or other people. Specifically, components which may use toxic compounds, like the LCD, must be examined as well as possible states of operation which may threaten the user in some way, such as battery overcharge.

6.2 Ethical Impact Analysis

To evaluate the challenges of bringing a design to market, one must have a code of ethics with which to evaluate. For simplicity, the IEEE Code of Ethics will be used. The most relevant points in this code to the marketing process of our device include to "accept responsibility in making decisions consistent with the safety, health and welfare of the public, and to disclose promptly factors that might endanger the public or the environment," "to be honest and realistic in stating claims or estimates based on available data," and "to acknowledge and correct errors" [2].

The aim of maximizing welfare and limiting endangerment is conducive to such practices as addition of warning labels and inclusion of safety mechanisms. Under normal operation, the FLACtrac poses no threat to the user; however, certain accommodations should be made for possible malfunction, damage, or misuse. The only real malfunction that could possibly cause harm to the user would be that of battery overcharge. To eliminate the likelihood of this occurring, a mechanism to detect and/or prevent

overcharge should be added to the circuit, which would shut off power supply to the battery if such an event were to transpire. Furthermore, a warning label could be placed on the package delineating that if the device began to grow warmer than expected temperatures, the battery or device should probably inspected or replaced. In the event that the device is damaged, the main concern to the user would be the mercury in the LCD. Thus, before market, it would at least need a warning label advising against contact with the LCD should it break, or the LCD could be replaced with an organic LCD. Finally, the main misuse which could cause harm would be prolonged use of the device at high volumes, which could result in partial hearing loss at some point in the future. This, too, necessitates an advisory warning to the user.

Ensuring honest and realistic claims, in addition to correcting errors, implies the device should be extensively tested to establish that it functions as expected. This includes testing under various ambient air conditions as well as for prolonged use of the device. Likewise, software extremes should be investigated to confirm soundness of design, primarily the use of FLAC files at varying ends of the FLAC format spectrum since FLAC files are not of one standard format. These are matters of integrity and quality and are very important in bringing a device to market.

6.3 Environmental Impact Analysis

A special ethical consideration, namely the environmental impact of the device, must be taken into account as seriously as the earlier considerations if the product is to maintain integrity and intrinsic value. The point in the IEEE Code of Ethics regarding responsibility toward safety and welfare and disclosure of factors that might endanger the public or the environment can be applied broadly in this respect. As such, the green footprint of the device at each life cycle stage must be analyzed. Since the footprint of the FLACtrac is negligible during the stage of normal use, the manufacture and disposal stages will be those addressed.

The main environmental concern during the manufacturing stage of our device is the fabrication process involved in making the printed circuit board. This process invariably uses highly toxic chemicals, including cyanide-base compounds, however this is mostly

unavoidable. The environmental footprint during this stage could possibly be lessened by careful choice of manufacturer, though. For instance, Continental Circuits Corporation has put out information on how they are making the process "greener" by replacing certain aspects of fabrication with newer technologies [3].

To minimize the environmental impact at the disposal stage of our device, it would be fitting to try to ensure that as much of the device is as recyclable as possible, and that any unnecessary pollutants be eliminated. Since the FLACtrac uses a Lithium Ion battery, that particular component would not require special disposal according to the federal government [4]. Nonetheless it is still recyclable, and the user should be encouraged to do so, perhaps through use of the green recycle symbol and/or directive information in the user manual. The other prominent components of the device are the LCD and circuit board. The current LCD in the device contains mercury. Unfortunately it seems the primary methods for disposal are via the hazardous material handlers of landfills, incineration, or reuse in other products [5]. An alternative, though it would add to the expense of producing the FLACtrac, would be to replace the LCD currently being used with an organic LCD. The principle concern with respect to disposal of the circuit board itself is that it contains toxic compounds, and moreover, components are attached to the circuit via solder, which usually contains lead, a toxic compound. Fortunately, the recent development of lead-free solder can eliminate that particular pollutant. Lead-free solder has been shown to exacerbate some common problems in electronics, though these complications can now largely be avoided by using certain application methodologies [6]. Lastly, the circuit board itself can be expensive to dispose of directly. Instead it is better to send the boards to be handled and processed by companies which extract and reuse the raw materials [7]. All disposal information and recycle information should be included in the user manual.

6.4 Summary

In order to bring a quality product to market, considerations should always be given to the ethical and environmental impact of the product. To satisfy ethical considerations, the FLACtrac must be tested under extreme use conditions and a mechanism to protect the user from a state of battery overcharge must be added. To satisfy environmental

considerations, the economic feasibility of using lead-free and organic components when possible should be seriously investigated. Additionally, the product should come with appropriate user advisories for any components that may cause harm, like the battery or, perhaps, the LCD, and appropriate disposal information in the user manual.

7.0 Packaging Design Considerations

7.1 Introduction

As a portable audio device, the packaging of the FLACtrac is highly critical. For the product to be practical, it must be small enough to be handheld and light enough to be carried around. This section will analyze current digital audio players in the field to better grasp the challenges we face and gain insight on good practice in packaging, and then show the design used to meet the packaging challenges present in the FLACtrac.

7.2 Commercial Product Packaging

Fortunately, there are many examples of commercial digital audio players. As a highly competitive consumer market, packaging style and utility is of high importance in the marketability of a digital audio player. To help design the FLACtrac, we will look at three commercial product lines: the Apple iPod, Microsoft Zune, and the Sansa series by SanDisk. Table 7 compares the specifications of a few select devices from each product family.

7.2.1 Apple iPod Family

Though not the first consumer, portable digital audio player, the iPod line has captured an extremely high market share (as high as 82% in 2004) [1]. Apple Inc. released the original iPod in October 2001 as a hard-drive based, Macintosh-only device, but over the next few years a line of iPods was developed available with either a miniature hard drive or onboard Flash memory; and also began to support both Macs and PCs running Microsoft Windows, using proprietary software to load music onto the device.

The iPod is notable for its simple, minimalist user interface, which many find easy to use. All incarnations of the iPod (with the exception of the iPhone-like iPod Touch, which is more related to the touch-screen iPhone) are controlled primarily using a rotary wheel, originally mechanical on the first iPod design but capacitive-sensing design on later models. Mechanical buttons for "next track," "previous track," "home," and "play/pause" are placed under the capacitive wheel, so that the user can slide his or her finger around the wheel to make selections or push on the select button is then placed in the middle of the wheel. This, along with a relatively minimalist onscreen interface compared to other devices, sets the iPod apart from the competition and may be a key to its popularity.

Figure 2: iPod Classic [2]

There are only two major ports on most iPod models – a simple 3.5mm audio connection for headphones and a 30-pin proprietary dock connector; which facilitates FireWire and/or USB (depending on model); charging (often as a by-product of the FireWire/USB connection); line level audio; and on some models, video output. The resolution of the onboard LCD screen was a modest 160 by 128 pixels for the first iPod to 320 by 240 pixels on the current iPod classic model.

7.2.2 Microsoft Zune 80

The Microsoft Zune is a product family quite similar to the iPod. The Zune 80 is a much more complicated device than the iPods we examined. It has essentially all of the features of the full-sized iPods, plus an IEEE 802.11b/g wireless interface for connecting to the Internet and to other Zunes, as well as a broadcast-band FM radio. These features allow

the Zune to implement some advanced features. It can wirelessly share music with other Zunes, and purchase music from the online Zune Marketplace. Synchronization with the Windows PC can be accomplished wirelessly as well. Songs heard on FM radio can be purchased by automatically taking the Radio Data System (RDS) data and finding the matching song on the Marketplace.

Figure 4: Microsoft Zune [5]

The Zune interface consists of two buttons ("play/pause" and "back") and a capacitive touchpad known as the "Zune pad." Unlike the iPod click wheel, the Zune pad allows the user to make directional gestures in all four directions and tapping to select. This might make the Zune more intuitive than the iPod click wheel for some users, but it seems to be a matter of personal preference.

For the most part, other than the differences already mentioned the Zune is fairly similar to the iPod, although it might be marketed towards a more tech-savvy crowd who would take advantage of the network capabilities.

7.2.3 SanDisk Sansa

The SanDisk Sansa devices are different from the iPod and the Zune in that many models can use removable storage, in the form of microSD cards; which isn't exactly surprising since SanDisk is one of the largest manufacturers of flash memory cards. The microSD works with SanDisk's slotMusic system, which is an attempt to sell popular music on flash memory cards instead of

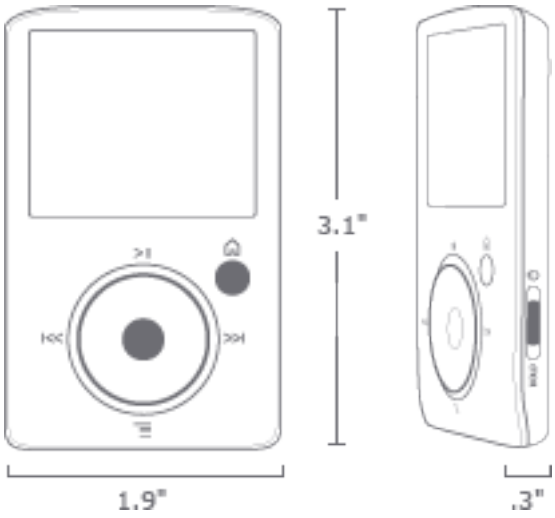


Figure 5: SanDisk Sansa Fuze [6]

compact discs or online downloading, allowing use of a small MP3 player without a computer.

The Sansa Fuze is the most inexpensive model that offers the microSD card slot. It is available for $79.99, $99.99, or $119.99 for 2, 4, or 8 GB models respectively. The Sansa Clip is a smaller, inexpensive player which uses internal flash memory, but is available for as little as $39.99 for the 1 GB model.

Both models have a user interface very similar to (and clearly inspired by) the iPod. The Fuze is smaller than the iPod classic and the Clip is even smaller than the iPod nano, already an impressively tiny device (although it is a bit thicker).

7.3 Project Packaging Specifications

The central concern in designing a digital audio player is making it small enough and light enough that it will be easy to carry. Table 7 shows the dimensions and mass of the FLACtrac and each of the commercial devices profiled. Some manufacturers take this to extremes – the iPod nano and the Sansa Clip are undoubtedly too small for many individuals. The biggest constraint in our packaging design is our LCD display, which is the largest component. Our desire to have a graphical LCD display (and the challenges of finding an appropriate device) dictate much of the resulting packaging.

	FLACtrac (projected)	Apple iPod (2001) [8]	Apple iPod classic [2]	Apple iPod nano [3]	Microsoft Zune 80 [5]	Sanza Fuze [6]
Height	127.4 mm	102.1 mm	103.5 mm	90.7 mm	108.2 mm	78.7 mm
Width	89 mm	61.7 mm	61.8 mm	38.7 mm	61.1 mm	48.3 mm
Thickness	25 mm	19.8 mm	10.5 mm	6.2 mm	12.9 mm	7.6 mm
Weight	256 g	185.9 g	140 g	36.8 g	128 g	59.5 g
Display	2.68” B/W 128x64 px	2” B/W 160x128 px	2.5” color 320x240 px	2” color 240x320 px	3.2” color 320x240 px	1.9” color 220x176 px
Cost	$100	$400	$250	$150-200	$230	$80-120

Table 7: Comparison of Various Commercially Available Digital Audio Players

Figure 6: Hammond 1553 Enclosure

In order to make a comfortable device to hold, we wanted to use a prefabricated enclosure that was designed for the purpose, if possible, since we can't afford to spend much time on ergonomic mechanical design. To achieve this we selected the Hammond Manufacturing 1553D enclosure (as pictured in Figure 6) because it appeared it would accommodate the LCD and lithium ion battery as well as a reasonably sized PCB. The results of squashing everything into this enclosure are shown in the drawings in Appendix B.

There are no requirements for spacing for heat dissipation listed in the data sheets for either our battery or the LCD, so we will have to wait until these parts are acquired to evaluate the effect of this on our packaging design. To maximize available space and the utility of the device, we plan to orient the LCD in a vertical fashion. Conventional buttons for controlling the device will be placed below the LCD screen. An SD card slot will be located in the bottom edge of the device.

7.4 PCB Footprint Layout

The datasheet for the Hammond enclosure dictates the optimal dimensions of the PCB it is designed to contain. This forms the biggest constraint to our PCB footprint. The basic dimensions of the PCB are 139.11 mm by 67.83 mm. A detailed drawing of the approximate layout and the exact notches and gaps required by the enclosure is also available in Appendix B.

7.5 Summary

The FLACtrac won't beat the iPod for any industrial design awards, or beat the Sanza products on weight or size, but then it's not really a fair comparison because large

production devices have the benefit of custom made components and mechanically engineered enclosures. Our packaging plan appears to be in the ball park of the state-of-the-art and should provide the end user with a sensible, capable device.

8.0 Schematic Design Considerations

8.1 Theory of Operation

The hardware of the FLACtrac can be divided into six major blocks. These six functional blocks are the pushbuttons, SD card, SHARC DSP, audio, LCD, and power blocks. The SD card and pushbuttons, are input blocks, while the LCD and audio blocks are output blocks.

The pushbutton block is very simple. It merely consists of four pushbuttons for user control of the device plus a reset button to reset the device if something goes wrong. The buttons interface directly with the DSP general purpose I/O pins. A resistor normally ties these pins high (+3.3V), but pressing the buttons tie the pin to ground. The reset pushbutton connects to each device with an "nReset" pin, including the SHARC DSP, LCD, and D/A.

The SD card block is also simple by virtue of the fact that the SD card can operate in a fully SPI compatible mode. Though high speed data transfers require a native, proprietary SD communications format, all cards are also required to support an SPI mode. The SPI mode is slower but fast enough for our requirements to load the FLAC data in real time. The SD card requires 3.3 volts for power and is compatible with 3.3 volt SPI, so it can connect directly to the SHARC DSP SPI bus.

These two input blocks feed into the core of the device, the SHARC DSP block. This block includes the Analog Devices ADSP-21262 SHARC DSP Chip itself, a 2Mbit Atmel SPI flash memory, and other supporting circuitry such as a crystal oscillator. The SHARC DSP has no onboard, nonvolatile memory, so the SPI flash is required for us to store our program, which is then loaded at boot time according to a pair of hardwired boot configuration pins. A 12.5 MHz quartz crystal is required for the SHARC's precision clock generator, which includes a PLL for clocking the DSP at 200 MHz and a clock

generator to be used for other purposes, namely driving the master clock on the AD1854 D/A in the audio block. The SHARC DSP will perform all of the computations to transform FLAC files into PCM data so it can be synthesized by the D/A. It also interfaces with the user via the pushbuttons and the LCD.

The audio output block takes care of turning the stereo PCM data output by the SHARC and outputting audio to a 3.5mm headphone jack. The SHARC outputs audio using an I^2S compatible serial port. The Analog Devices AD1854 D/A takes this serial data and converts it to a analog stereo output. The AD1854 is a sigma-delta type DAC and is capable of 44.1, 48, and 96 kHz sampling rates. The analog output of the AD1854 is then routed to an Analog Devices SSM2135 audio amplifier IC. This, along with some RC low-pass filtering forms the audio output circuitry.

The other major output block of the device is the LCD display. This block is comprised of the LCD module (Crystalfontz CFAG12864B-TMI-V) and supporting circuitry. The display is a 128 by 64 pixel monochrome display with an 8-bit parallel interface. However, because of the shortage of general purpose I/O pins on the SHARC DSP, we decided to use the SHARC's 16-pin parallel port in GPIO mode rather than as a parallel port. This meant that we had to design a serial-to-parallel interface. An SPI-compatible shift register (Texas Instruments 74HC595) meets this need. The LCD also operates on 5 volt logic levels, and is not 3.3V tolerant (there is an insufficient noise margin), so we also added a logic level converter (Texas Instruments CD4504B). A trimmer potentiometer for the contrast is located on the PCB, as well as a resistor to limit current flow to the LED backlight.

Lastly, the power block supplies the 1.2, 3.3, and 5 volt power rails required by all of the previous blocks. It also contains a coulomb counter which provides current flow information to the SHARC DSP for the purposes of informing the user with a battery "fuel gauge." It has the capability of running off of an internal lithium ion battery or a "wall wart" style AC adapter, which also charges the battery if it is depleted. The LCD and the audio blocks both require a 5 volt power supply, which requires us to use a charge pump to boost the voltage. The SHARC DSP requires a 1.2 volt supply for the processor core,

as well as 3.3 volt for the peripherals. All other blocks only require a 3.3 volt power supply.

8.2 Hardware Design Narrative

Though the ADSP-21262 DSP is capable of large amounts of data processing, it is not as capable in terms of having a large number of peripheral pins. In fact, the SHARC comes with only four general I/O pins in its default configuration. Fortunately, we can disable the parallel port in order to gain access to 16 additional I/O pins. Fourteen of these pins have been allocated to communication among the blocks of the device. The DSP also has an SPI bus (capable of being a master to four slaves, up to 16 slaves if you bit-bang the slave select lines to the parallel port GPIO) and has an additional 20 pins devoted for use as a digital audio interface. An internal signal routing unit allows these twenty pins to be connected in various low-latency ways to six full-duplex internal serial ports designed for high speed audio communication (unfortunately, these audio serial ports are not general purpose, they do not form a UART). In our implementation, we utilize the parallel port pins as general I/O pins as mentioned previously, the SPI module as a master with 3 different slave devices (LCD, SD Card, and SPI Flash for program storage), and one of the serial ports available on the digital audio interface for the AD1854 D/A. We chose to disable the parallel port because we required more general purpose I/O pins than the DSP would otherwise provide. These pins are required to provide interfaces with the coulomb counter, act as slave selects for the SPI bus and provide additional clocking and miscellaneous signals to the LCD Controller. The SPI module was selected, particularly for the SD Card interface, for its ease of use and widespread availability. We chose to interface with the AD1854 D/A by using the I^2S serial because it seemed to be the most compatible transmission standard. In general, the only configuration choices we had to make as far as other subsystems are concerned is the choice to use an SPI to parallel shift register to interface with the LCD controller. The reason for this decision is quite simple; we would not have had enough pins on the chip to complete the project had we not. For this particular system, the complexity is not increased significantly in hardware, but rather the primary complexity arises in software, since both the SPI module and the general I/O pins have to be used.

8.3 Summary

In order to complete our project and satisfy our Project Specific Success Criteria, we must implement six main blocks. These blocks are primarily input and output blocks, as the fundamental property of our device in interaction with a user. Our primary input block, the SD Card Controller, will read data from attached storage and send it to our only processing block, the DSP itself. Here, the DSP chip will decompress the data and feed it back to the user through our primary output block, the audio D/A. In addition to the three core blocks, the pushbutton block exists to control the playback of audio, the power block naturally exists to provide power to the device, and the LCD block exists to display information to the user in an easy-to-understand format. The interfaces within and between these blocks are well defined and follow standard interface specifications.

9.0 PCB Layout Design Considerations

9.1 Introduction

The FLACtrac needs to be packaged in a small case to minimize size and make it into a portable device that will fit in a pocket. Because of this, the PCB layout is extra challenging. The components have to be placed around the physical constraints that the Hammond 1553 case and the dimensions of the larger components present. Also, as an audio device, we especially want to cut down to cut down on EMI noise, designing our placement in a way that the audio signals are not affected by the digital circuitry.

9.2 PCB Layout Design Considerations - Overall

Our layout is largely determined by the space allowed by the enclosure that we have chosen. Our large parts include the external interfaces (SD Socket, Power connector, and 3.5mm headphone jack), the battery, and the LCD. As mentioned previously in Section 7.3, the best placement places the LCD and battery on top of each other leaving a small clearance between that and the PCB. This requires us to place any components on the back side of the PCB that would lie in the area where the battery is.

The rest of the large components are the external connectors. We have placed the headphone jack on the top of the board. Because of this, we have moved the LCD and battery up toward the top till it is close to that of the headphone jack. Because of this our

audio components will be on the backside of the board underneath the battery. This leaves area on the lower portion of the PCB to be used for the other external interfaces. The SD socket and power connectors are placed along the bottom edge.

Thus the board is divided up into 3 regions that help eliminate EMI between the different areas. First is the digital region. It spans most of the left side of the board leaving some room at the top for another section. Going along the right edge is the power circuitry. The power and battery connectors will connect in this area and run to the other components. Last is the audio region. It occupies the top part of the board. It receives the digital audio signal into the DAC and the rest is analog to the headphone jack.

Although we intend to follow these divisions, we might have to expand some of the power circuit into the digital area. There are a few larger components the may require space that is free between the battery/LCD and that of the SD card and power connector. This space is currently divided between digital and power, but the power might poke into the digital region because of some larger than suspected inductors and capacitors present into the battery/charging circuit.

9.3 PCB Layout Design Considerations - Microcontroller

The DSP that we are using requires bypass capacitors to be placed between the power and ground of the input very close to the pins into the chip. The DSP has two voltage sources, one at 1.2 V and the other at 3.3 V. Because of this, we have to have two separate sets of bypass capacitors. We have to get the capacitors as close to the chip as possible. To do this, we are using a larger capacitor and a lot of smaller ones. Because of the spatial concerns that are presented by the battery and LCD screen, the large capacitor will go on the same side as the DSP and the small capacitors will go on the other side in the small space between the PCB and the battery directly underneath the DSP.

Within the digital section of the layout, all the major components communicate with the DSP. Because of this, the chip has to be centrally located in that block. All of the digital chips communicate with the DSP via the SPI interface. So we placed the DSP in the middle of everything to give us optimal placement and routing to the other components.

With optimal routing, we reduce required vias and reduce possible EMI interference that might show up on an non-optimally routed line.

9.4 PCB Layout Design Considerations - Power Supply

Our power supply is placed in the bottom right corner and the traces will go up the right side of the board. The power connector and battery will connect into this area where it will branch to the three regulators. From these regulators, the traces will split and go to the separate areas. All of the 1.2 V and 3.3 V lines will only go to the digital circuitry. The 5 V line requires a little more caution in placing. The LCD and associated shift register run off of 5 V as well as the DAC and all of the associated circuitry. To reduce coupling as much as possible, we will split these traces as close to the regulator as possible. From there we can use the two as a digital 5 V line and an analog 5 V line. This will reduce coupling and make our circuit work better.

9.5 Summary

Our PCB layout is restricted mainly by packaging constraints. We have three external connectors that must be placed on the edge of the board. We also have the battery and LCD that are very large. These will be placed in the middle with the LCD on top of the battery. The rest of the layout will be divided into three regions: audio, power, and digital. The audio region is on the top of the board, the power section is only the right side of the board, and the digital region is on the left. Using this method of division, we can reduce coupling and make our circuit perform better. We are using coupling capacitors to fit the specifications that the DSP asks for, placing larger ones on the same side as the DSP and small capacitors under the DSP between the PCB and battery. Finally we will use separate traces for each section originating from the power section. This will reduce coupling between the different circuits and allow our device to perform the necessary requirements.

10.0 Software Design Considerations

10.1 Introduction

The most important consideration for the FLACtrac design is time. It has to process data from FLAC files in real time, so that the listener can hear their lossless music

continuously, otherwise it would not be a very useful device. This pressing requirement has led to several development strategies regarding computational efficiency.

First, the inherent parallelism of the DSP must be utilized to facilitate best performance. The SHARC DSP has two processing elements, and as such, it can be set to use only one element, called single instruction single data (SISD) mode, or both elements, called single instruction multiple data (SIMD) mode [2]. SISD implements the given instruction on one piece of data, while SIMD implements the given instruction on two pieces of data [2]. Since the memory is dual-ported, and the core contains an instruction cache, the instruction can be pulled from cache while a piece of data is sent over each bus. This suits the repetitive audio algorithm used in the FLACtrac very well, as now twice the work can be done with one instruction. To enable PEY, the second processing element, the PEYEN mode bit in the MODE1 register will be set [1].

Furthermore, the RAM can be organized in such a way as to even more efficiently utilize the parallel capabilities embodied by the use of SIMD mode. The DSP has two separate dual-ported memory blocks, each with 1Mb of SRAM, located in IO processor registers 0x0000 0000–0003 FFFF [1]. To minimize overhead and bus conflicts, one block will store data and use the Data Memory Bus and the other will store both instructions and data and use the Program Memory Bus.

Another development consideration revolves not around the real time constraint but around an earlier design decision to disable the parallel port and use it for general purpose IO. Without disabling the parallel port, there would only have been four flag pins available, which would not have been sufficient [1]. However, the port was not strictly necessary, so it will be disabled by setting bit 20 of SYSCTL [1]. Every peripheral the DSP interfaces with, except the digital to analog converter, will use these pins, which can be accessed directly via the IO processor registers.

This leads to the determination of the overall code organization of the FLACtrac. The real time constraint provides motivation to keep the design as event-driven as possible to minimize non-productive code. Nonetheless, the decision to change the parallel port to general purpose I/O had a side effect in that it limited the number of interrupts that could

be sustained, thereby dropping the number of available interrupts under the number needed to keep user pushbuttons entirely interrupt-based. Instead, the buttons will be polled, and flags set, according to an internal timer interrupt, approximately every 10 ms. The LCD will also be updated on/ this timer. The use of the timer should not be too much of a drawback though, as the lack of priority for these functions will be reflected in the timer's low frequency of interrupts. The coulomb counter will be maintained as an external interrupt.

Software programming and debugging will occur via a JTAG header on our PCB, and with the Analog Devices supplied development environment, VisualDSP++.

10.2 Software Design Narrative

The most important and repetitive software will be implemented in the main module. This includes, starting DMA at the correct times, both the DMA from the SD card to the in-buffer and the DMA from the out-buffer to the DAI, and checking pushbutton flags. However, most significantly this includes decoding the FLAC data from the in-buffer and placing it in the out-buffer and, additionally, computing the visualization information. This module is the most computationally intensive. It is partially implemented, though none of it has yet been tested.

The timer interrupt module will include two basic sub-functions, polling the user buttons on pins AD3-6, and updating the LCD. Updating the LCD includes populating the pixels with the visualization information, the battery life information, and the song information. All of this data will be stored before accessed by the LCD updating function; therefore it will be a fairly simple population. From there, the data will be communicated over SPI to the 74HC595 shift register, and shifted out serially to the LCD.

The Coulomb counter will be serviced based upon an external interrupt through pin AD2. When triggered, the service routine will read which direction the charge is traveling based upon pin AD1 which contains the polarity of the current as dictated by the Coulomb counter. The charge count will be updated accordingly.

Currently, no shareware or other code segments appear necessary for developing the software, other than the examples provided by VisualDSP++. The examples are simply to get an idea of how certain aspects of the design might be laid out or how certain aspects should interact with each other in implementation.

10.3 Summary

The application design for the FLACtrac essentially revolves around real time considerations, considerations, utilization of the parallelism in the DSP, and previous design decisions. The code organization is as event-driven as possible due to the real time constraint, though due to design decisions, the user pushbuttons will have to be polled. The parallelism in the DSP is rather feasibly employed by setting the processor mode to single instruction multiple data mode, wherein, one instruction executes on two pieces of data, one from each block of RAM. The parallelism can be abetted by properly structuring the RAM.

The overall structure of the software modules relies on a main loop which does the decoding and computation. This loop is interrupted for the Coulomb counter on external interrupt, LCD update and pushbutton polling on timer interrupt, and DMA transfer on internal interrupt.

The complete listing of all of our C source code is listed in Appendix F.

11.0 Version 2 Changes

There are several hardware issues, which if addressed in a new PCB revision could significantly improve the performance and cost of the FLACtrac. The originally specified AME8890 1.2V regulator was completely inadequate for powering the SHARC DSP (which requires over 500 mA at 1.2V), so the PCB needs to be changed to accommodate an appropriately-sized regulator. The footprint of the SD card socket was also slightly incorrect and resulted in some creativity being necessary to make the right connections.

There are also component alternatives which could significantly lower the cost, complexity, and performance of the product. A certain OLED module is now available from Crystalfontz which can be controlled directly using SPI (and thus our 74HC595

would be unnecessary) and the CD4504 logic level converters could be eliminated from the LCD circuitry as well. In retrospect, the choice of the SHARC DSP itself was probably a poor one in terms of battery life. The device pulls around 600 mA of current at the nominal battery voltage of 3.7 V, which results in an anemic battery life of only approximately three hours. The primary consumer of this power is the SHARC itself, although it is possible that further tuning of the power management circuitry could increase efficiency as well. The device currently has no real "off" or "sleep" state, which needs to be remedied for a production device. A transistor circuit could be used to let the SHARC (or a more power efficient processor) control the LCD backlight LEDs, perhaps at several levels of brightness. A transistor circuit could be designed utilizing the "card inserted" detection pins on the SD socket to power on the device only when an SD card is inserted.

12.0 Summary and Conclusions

Our group was able to implement four of our five PSSCs, with very nearly the fifth PSSC completed. Issues with increasing the baud rate of the SPI bus and still maintaining stable communication prevented us from being able to decode FLAC files at their full speed. We feel confident that we could implement solutions to nearly all of the outstanding problems to make the FLACtrac a production quality device. The group learned how to communicate with an SD card, implemented a read-only FAT file system driver, much about audio hardware and software, and about designing power supplies and interfacing components. No one on the team had any previous experience with the Analog Devices SHARC family, and we managed to have good success with learning the architecture on our own without any outside help.

Our senior design project was highly successful, both in the device we were able to implement and more importantly in the lessons and skills we learned for the future. We have substantially increased our knowledge and honed our skills at developing solutions for real-world applications.

13.0 References

13.1 Section 3.0: Constraint Analysis and Component Selection

[1] SD Group, "SD Specifications, Part 1; Physical Layer Simplified Specification," Sep. 25, 2006.

[2] "Apple – iPod nano," apple.com, Sep. 9, 2008. [Online]. Available: http://www.apple.com/ipodnano. [Accessed: Feb. 4, 2009].

[3] "Analog Devices: ADSP-21xx Processors – Embedded Processing and DSP," analog.com. [Online]. Available: http://www.analog.com/en/embedded-processing-dsp/adsp-21xx/content/index.html. [Accessed: Feb 4, 2009].

[4] "Analog Devices: SHARC Processors – Embedded Processing and DSP," analog.com. [Online]. Available: http://www.analog.com/en/embedded-processing-dsp/sharc/content/index.html. [Accessed: Feb 4, 2009].

[5] "Freescale DSP563xx Series Digital Signal Processors," freescale.com. [Online]. Available: http://www.freescale.com/webapp/sps/site/taxonomy.jsp?nodeId=0127958596. [Accessed: Feb 4, 2009].

13.2 Section 4.0: Patent Liability Analysis

[1] Takezawa, Masayuki, Recording and Reproducing Apparatus which Calculates and Displays Management Information of Recorded Segments, Tokyo, Japan: 1995. [.pdf] Available: http://www.minidisc.org/patents/pdfs/US05392265.pdf

[2] K. Tairo, H. Mimura, Recording/Reproduction System of Music Data, and Music Data Storage Medium, Tokyo, Japan: 2003 [.pdf] Available: http://www.google.com/patents/download/Recording_reproduction_system_of_music_d.pdf?id=1zsOAAAAEBAJ&output=pdf&sig=ACfU3U1U1lAg8ChHPkREqfe6u_p-JRNqaA

[3] D. Diamond, D. Glowny, T. Nguyen, P. Min Ni, and J. Richter, System and Method for Data Recording and Playback, Southbury, CT: 2001 [.pdf] Available: http://www.google.com/patents/download/System_and_method_for_data_recording_and.pdf?id=GLkIAAAAEBAJ&output=pdf&sig=ACfU3U39DDKlTQQYezzglzoJiyyAyGE7QQ

13.3 Section 5.0: Reliability and Safety Analysis

[1] Department of Defense, "Military Handbook Predicting Reliability of Electronic Equipment (Mil-Hdbk-217F)," [Online Document], 1991 December 2.

[2] Analog Devices, "SHARC Embedded Processor," ADSP-21262 datasheet, July 2008.

[3] Linear Technology, "Standalone Linear Li-Ion Battery Charger with Thermal Regulation in ThinSOT," LTC4054B datasheet, 2003.

[4] Texas Instruments, "High Efficiency Single Inductor Buck-Boost Converter with 1-A Switches," TPS63030 datasheet, October 2008.

[5] Linear Technology, "Micropower High Output Current Step-Up Adjustable and Fixed 5V DC/DC Converters," LT1302 datasheet, 1995.

13.4 Section 6.0: Ethical and Environmental Impact Analysis

[1] F.G. Splitt, "Environmentally Smart Engineering Education: A Brief on a Paradigm in Progress," *Engineering Education Reform: A Trilogy,* pp. 1-4, January 2003.

[2] IEEE, "IEEE Code of Ethics," *IEEE,* Policy 7.8, February 2006. [Online]. Available: http://www.ieee.org/portal/pages/iportals/aboutus/ethics/code.html. [Accessed: April 16, 2009].

[3] Continental Circuit Corporation, "Making High Quality Circuit Boards with Fewer Toxic Chemicals," *CCC,* September 1994. [Online]. Available: http://www.p2pays.org/ref/03/02120.pdf. [Accessed: April 16, 2009].

[4] Panasonic Corporation, "Battery Disposal Guidelines," *Panasonic,* 2009. [Online]. Available: http://www.panasonic.com/industrial/battery/oem/enviro/index.html. [Accessed: April 16, 2009].

[5] The A to Z of Materials, "Recycling Liquid Crystals from Waste LCD Devices," *AZoM,* September 2006. [Online]. Available: http://www.azom.com/news.asp?newsID=6560. [Accessed: April 16, 2009].

[6] M. Goosey, "Soldering considerations for lead-free printed circuit board assembly," Envirowise. [Online]. Available: http://www.emeraldinsight.com/Insight/ViewContentServlet?contentType=Article&Filename=Published/EmeraldFullTextArticle/Articles/2170310307.html. [Accessed: April 16, 2009].

[7] Joint Service Pollution Prevention, "Printed Circuit Board Recycling," *Joint Service Pollution Prevention Opportunity Handbook,* May 2003. [Online]. Available: http://205.153.241.230/P2_Opportunity_Handbook/2_II_8.html. [Accessed: April 16, 2009].

13.5 Section 7.0: Packaging Design Considerations

[1] "Apple's Jobs Taps Teen iPod Demand to Fuel Sales, Stock Surge," bloomberg.com, Oct. 11, 2004. [Online]. Available: http://www.bloomberg.com/apps/news?pid=10000103&sid=a58iozj_2jXM. [Accessed: Feb. 12, 2009].

[2] "Apple – iPod classic," apple.com, Sep. 5, 2007. [Online]. Available: http://www.apple.com/ipodclassic. [Accessed: Feb. 4, 2009].

[3] "Apple – iPod nano," apple.com, Sep. 9, 2008. [Online]. Available: http://www.apple.com/ipodnano. [Accessed: Feb. 4, 2009].

[4] "How iPods Work," howstμFfworks.com, Mar. 14, 2006. [Online] Available: **http://electronics.howstuffworks.com/ipod4.htm**. [Accessed: Feb. 12, 2009].

[5] "Zune.net – Zune 80." zune.net, Jun. 13, 2008. [Online] Available: http://www.zune.net/en-us/mp3players/zune80/default.htm. [Accessed: Feb. 12, 2009].

[6] "Sansa," sansa.com, Mar. 28, 2008. [Online] Available: http://www.sansa.com [Accessed: Feb. 12, 2009].

[7] "Hammond Mfg. – Soft Sided Hand Held Enclosures (1553 Series)," hammondmfg.com. [Online] Available: http://www.hammondmfg.com/1553.htm [Accessed: Feb. 12, 2009].

[8] "iPod," apple-history.com, Apr. 6, 2003. [Online] Available: http://www.apple-history.com/?page=gallery&model=ipod&sort=date&performa=off&order=ASC [Accessed: Apr 22, 2009].

13.6 Section 8.0: Schematic Design Considerations

[1] Analog Devices, "ADSP-21261/ADSP-21262/ADSP-21266 SHARC Processor Data Sheet (Rev. E)", 2009 [Online]. Available: http://www.analog.com/static/imported-files/data_sheets/ADSP-21261_21262_21266.pdf [Accessed February 19, 2009]

[2] Crystalfontz, "Graphic LCD Module Specifications", 2009 [Online]. Available: http://www.crystalfontz.com/products/12864b/datasheets/1152/CFAG12864BTFHV_v2.0.pdf [Accessed February 19, 2009]

[3] Analog Devices, "AD1854", 2009 [Online]. Available: http://www.analog.com/static/imported-files/data_sheets/AD1854.pdf [Accessed February 19, 2009]

[4] Atmel, "2Mbit High Speed SPI Serial Flash Memory", 2009 [Online]. Available: www.atmel.com/dyn/resources/prod_documents/doc2455.pdf [Accessed February 19, 2009]

[5] Linear Technology, "Coulomb Counter / Battery Gas Gauge", 2009 [Online]. Available: http://www.linear.com/pc/downloadDocument.do?navId=H0,C1,C1003,C1037,C1134,P2354,D1556 [Accessed February 19, 2009]

13.7 Section 9.0: PCB Layout Design Considerations

[1] M. Glenewinkel, "System Design and Layout Techniques for Noise Reduction in MCU-Based Systems," CSIC Applications, Austin, TX.

13.8 Section 10.0: Software Design Considerations

[1] Analog Devices, "ADSP-21261/ADSP-21262/ADSP-21266 SHARC Processor Data Sheet (Rev. E)", 2009 [Online]. Available: http://www.analog.com/static/imported-files/data_sheets/ADSP-21261_21262_21266.pdf [Accessed March 25, 2009]

[2] Analog Devices, "ADSP-2126x SHARC DSP Core Manual (Rev. 2)", 2004 [Online]. Available: http://www.analog.com/static/imported-files/processor_manuals/485684862062146484521 26x_HRM.pdf [Accessed March 25, 2009]

Appendix A: Individual Contributions

A.1 Contributions of Greg McCoy:

We started the semester with an idea of doing a MicroMouse robot, but no one on the team felt strongly about it. After Dr. Johnson announced that there was sponsorship available for projects based on implementing a FLAC audio device, we decided to do that instead because it seemed more feasible with our skills and applicable to our own interests.

As we surveyed the interests and proficiencies of each member of our group, it seemed that I was most interested in working on the hardware, so I made this my primary focus. I took responsibility for the early homework assignments, doing the Design Constraint Analysis and the Packaging Design. Since we were attempting to make a handheld device, I felt that the component selection and the packaging design was extremely critical to the success of our device. I used 3D CAD modeling software (Autodesk Inventor) to help guide my component selection, choosing LCDs and a battery which would fit within our enclosure, and choosing components that were short enough to fit in the 13 mm on the "tall" side and the 4mm on the "short" side. As a group we looked through various DSPs for the microcontroller selection, and Isaac took care of the power supply circuitry, but I took the lead on all other parts selection, and maintained the Bill of Materials throughout the semester. I also ordered all of our parts, mostly working with our sponsor, Mark Brooks of the Southwest Research Institute, but also purchasing some development tools on my own. I spent time (along with Brett) setting up and learning how to use the Analog Devices development tools and evaluation boards available in lab to get hardware design testing and software development going.

I then assisted Isaac with the initial schematic and Brett with the PCB layout. During this period, I helped Isaac and Brett a lot with circuit design. We initially had a very poor understanding of the capabilities of our device, and by experimenting with the evaluation board and spending a lot of time reading documentation, I helped to improve our circuit design quite a bit. I designed and selected the parts for the LCD and the logic level

converters, which were necessary to interface the SHARC DSP with the 5 volt LCD screen. I also selected the D/A converter and audio output circuitry. We struggled with PCB layout for several weeks because of the severe size constraints on our PCB size, and we had issues with drilled holes being incorrect which resulted in our PCB not being fabricated over spring break. Fortunately, once the PCB layout was accepted after spring break it took just under a week for the board to arrive. During this waiting time, I also set up "test stands" for the SD card and the LCD with two of the evaluation boards from Analog Devices while we were waiting on our PCB, which helped to expedite the software development by Brett and Isaac, which they could do in parallel with my work on populating the circuit board. First, I added the power circuitry and tested it, which seemed stable, and then added the SHARC DSP. I discovered that the 1.2V regulator we selected did not supply enough current for the DSP, so I talked to Karl and fly-wired an LDO he gave me to make the DSP run. I spent approximately 40 hours on PCB population and testing alone. I practiced soldering the LQFP devices and it didn't take long before I felt very comfortable doing it and I really enjoyed soldering the entire board – especially when we got it to work!

I also helped some with software development, but I was unable to help to the extent that Brett and Isaac did because they had made a lot of progress on software while I was working on assembling the circuit board and I was somewhat "out of the loop" on our software design. Still, I assisted as much as I could by discussing concepts with them and providing information about the hardware.

A.2 Contributions of Brett Mravec:

I started the semester researching the flac audio format. Using this information and the reference implementation, I created my own implementation of a flac decoder setup to run on a normal computer system. Later in the semester, I modified the implementation after we decided on using the sharc and set it up as a state based decoder allowing the ease of decoding frame by frame while still checking the buttons and updating the lcd inbetween.

After the initial implementation of flac, I moved on to working on the layout of the board. There were several major components that required a custom footprint to be made in

OrCAD, most notable was the SD card socket. These custom footprints were then loaded and used to layout the entire board. I completed a majority of the layout with the assistance of Isaac and Greg. We had several problems with out layout after it was sent to fabrication but eventually we fixed these errors and got it made successfully. During this time I also completed the associated homework on layout. At a later time I also completed the reliability and safety paper.

After this I moved to using a dev board to write the code to communicate to the SD card. Greg got a SD socket on a breakout board and connected it via SPI jumpers on the dev board. Later I determined that the current way I had setup the code was not reliable enough to successfully decode flac files so I implemented crc checking to allow for error detection. This greatly improved the reliability of the data being read and made decoding possible. However the communication link was not very good and gave a large amount of crc errors which slowed down our transfer rate substantially.

Once I finished off the initial implementation of the SD card routines, I transitioned into implementing a read-only fat filesystem. Our device only needs to read and parse flac files so there is no need to actually write to the device. To implement this fully requires a large amount of memory, so to work correctly I implemented a sort of cache that kept pertinent portions of data in memory only loading data when it is needed.

Once I got all of the file access capabilities working I moved to working with audio. We have limited memory available on the chip itself so we were forced to have several small audio buffers. That way we can effectively use DMA and still have space to load the buffers. I set it up as a circular array of buffers. This allows for the most effective use of the DMA system and limited memory.

After this step the systems were ready to be integrated into a final program that can load and play flac files. I setup the main loop to use a state machine and keep track of the decoder being used to decode the currently opened file. This would allow for other codecs to be implemented in the future. The state system allowed for an easy system to control what is being displayed on the screen and how the system reacts to user input (buttons).

This left the system in a complete state but it included a large number of bugs. I then moved into the debugging stage fixing a majority of the problems that were present in the code. I also did some coding to fix and simplify several routines. I modified the code to include much more error checking than was initially present. This made the system more reliable, much more able to cope with error conditions, and more able to inform the user of what types of errors have occurred.

A.3 Contributions of Isaac Jones:

Prior to the beginning of the semester, we met as a group to discuss a project that we did not end up using. During this time that would normally be devoted to research concerning the project, I did research concerning the project that we did not end up implementing. During the first TCSP, we were told about an opportunity to have our project sponsored by Southwest Research Institute. The only caveat for this requirement was that we implement Rice Compression in some way. Since we could not think of a way to incorporate Rice Compression in the original MicroMouse project, we chose to implement a FLAC-based audio player instead. I spearheaded the effort to contact Professor Johnson to change our project and then have it sponsored by SRI.

Early in the semester, Greg and I were the primary contributors to the parts selection of the DSP and other components that we would ultimately use for the project. Though our initial design called for both a microcontroller and a DSP, we eventually decided that using a very fast DSP would be sufficient for our needs. After I analyzed the data processing requirements, I felt that having only the DSP would be sufficient for our needs. As the hardware was finalized, Brett and I continued to work on the high-level software design. This would allow us to have an easy framework to follow when actually programming the device.

As I was responsible for the Preliminary Schematic, the responsibility fell on me to do much of the design as well. In particular, I designed, with TA assistance, the power circuitry and the circuitry that interfaced between the various voltage levels of the circuit. During this process, we discovered that there are quite a few intermediate parts we need to added to the circuit, so research was done to find parts that met those particular criteria. In

particular, I conducted the research for, designed the circuit for, and integrated the Coulomb Counter into our design. At this time, I also completed the Preliminary Schematic Homework assignment, which included a theory of operation document. This theory of operation is now a little outdated, as software constraints have forced us to change this operation.

After the Preliminary Schematic was complete, the group moved on to the PCB design. I was not particularly involved in the PCB design during the first week, but during the second week of PCB design, I was more active. During the first week of PCB design, we changed a few things on the actual Schematic for compatibility with the layout software. As we left for spring break, we finally completed the PCB design. After spring break, we worked on correcting the errors that appeared in our design, which included correcting the footprint for some of the parts I had advocated.

After finally submitting the Preliminary Schematic, I dove headfirst into the LCD. Since the documentation on the LCD was lacking at best and in Chinese for the most part. For this reason, most of the work I did with the LCD consisted of sending random command to the screen trying to eke functionality out of it. Unfortunately, we discovered that the LCD screen we were trying to work with was non-functional due to an error in hooking up the contrast rail of the LCD. After replacing the LCD module, we were able to correctly interface with the LCD. During this process, I wrote the Patent Liability Analysis homework. After the completion of this homework, I wrote a first draft of the Audio Codec interface for the DSP. Though it was later overwritten by Brett for efficiency reasons, much of the fundamental work to communicate via I^2S and the implementation of DMA on the SPORT is my code. After the completion of this, I have been assisting Brett with de-bugging the FLAC code in an effort to gain full functionality.

A.4 Contributions of Danielle Miller:

My role on the team was largely a support role with a focus on software organization. This is partially due to my late arrival to the team, with most members already having delineated their desired function and focus for the project. I aided with fundamental design choices, helped out with PCB routing, drove software layout, and assisted with

debugging. Since I majored in computer engineering, more of my concentration was drawn toward software and higher level design decisions when it came to hardware. With respect to project documentation, I provided the software narrative and the ethical and environment impact analysis, as well as participating in the group-created documentation.

There were several design decisions that contributed greatly to the flow of development of our project. The main decision affecting the course of design was choosing a microprocessor. I tried to point out advantages and disadvantages in lieu of narrowing down the selection and tried to direct attention to possible problems with each, especially regarding memory, clocking, and power consumption. I was adamant about selecting a component with the maximum memory and clock frequency that would be feasible to utilize since these would surely be heavy constraints for our device. Decoding and outputting FLAC files would be very memory intensive and require high throughput. Also, because this project was the first of its caliber for most of us, and had to be completed within a limited amount of time, I tried to stress erring on the side of caution, meaning that extra resources would be better than an insufficiency of resources.

When it came to software I helped to lay a lot of the groundwork, composing pseudo-code, constructing a design hierarchy, and researching modes of operation and structure for the DSP. I outlined the basic modules we would need and how they would fit into the whole. I tried to direct team discussion to address software with relation to previous design decisions and to weigh benefits of different methods and how practical those methods would be to implement. After considering certain design constraints and real time motivation, the resultant design flow was of a hybrid nature with a mixture of polling and interrupts. Additionally, I researched how to exploit the inherent parallelism of the DSP via setting modes for the processing elements. I discovered, furthermore, that the RAM can be structured to facilitate further parallelism by generally organizing each block to use a separate bus and, as much as possible, separate data types. This helps overhead and bus conflicts.

Other than these contributions, I aided where teammates requested assistance. This includes things from trying to puzzle out some PCB routing to enduring some software/ hardware debugging, as well as writing some code for interfacing with the LCD.

Appendix B: Packaging

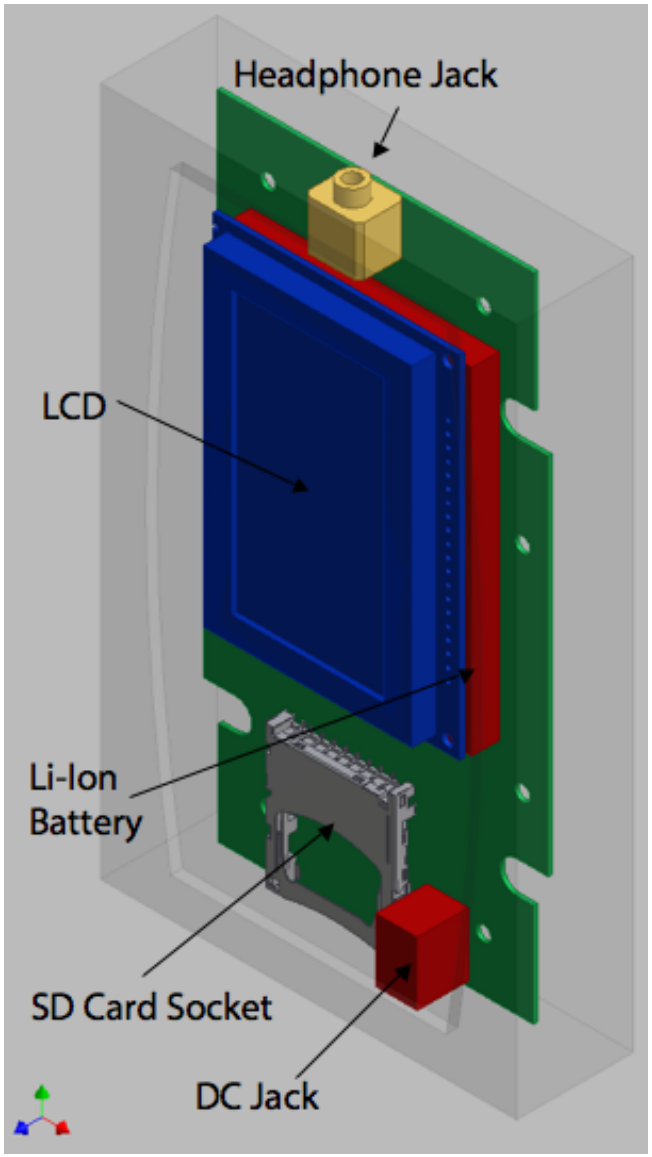


Figure B-1: Annotated Internal Diagram

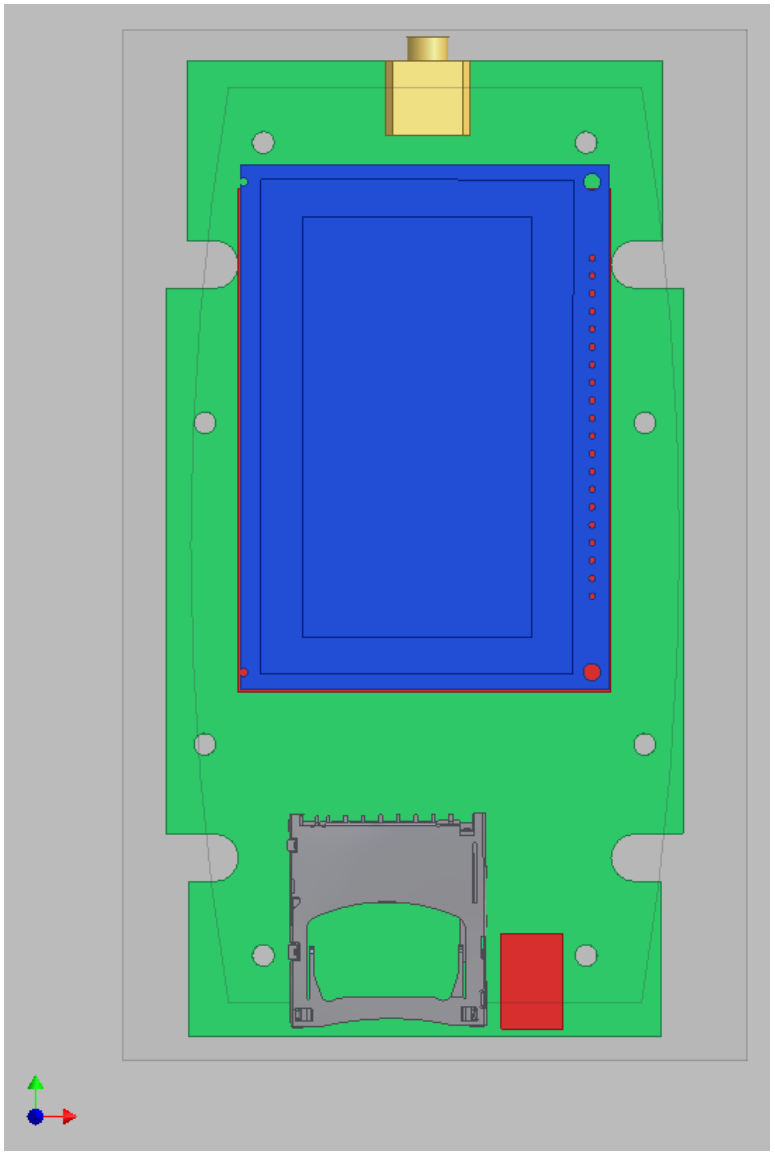


Figure B-2: Top View

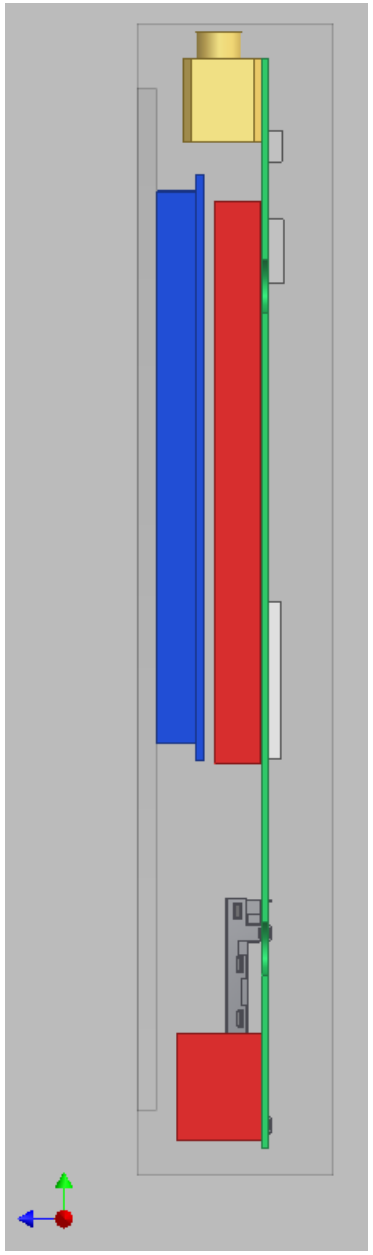


Figure B-3: Side View

Figure B-4: Photograph of Complete Packaged FLACtrac

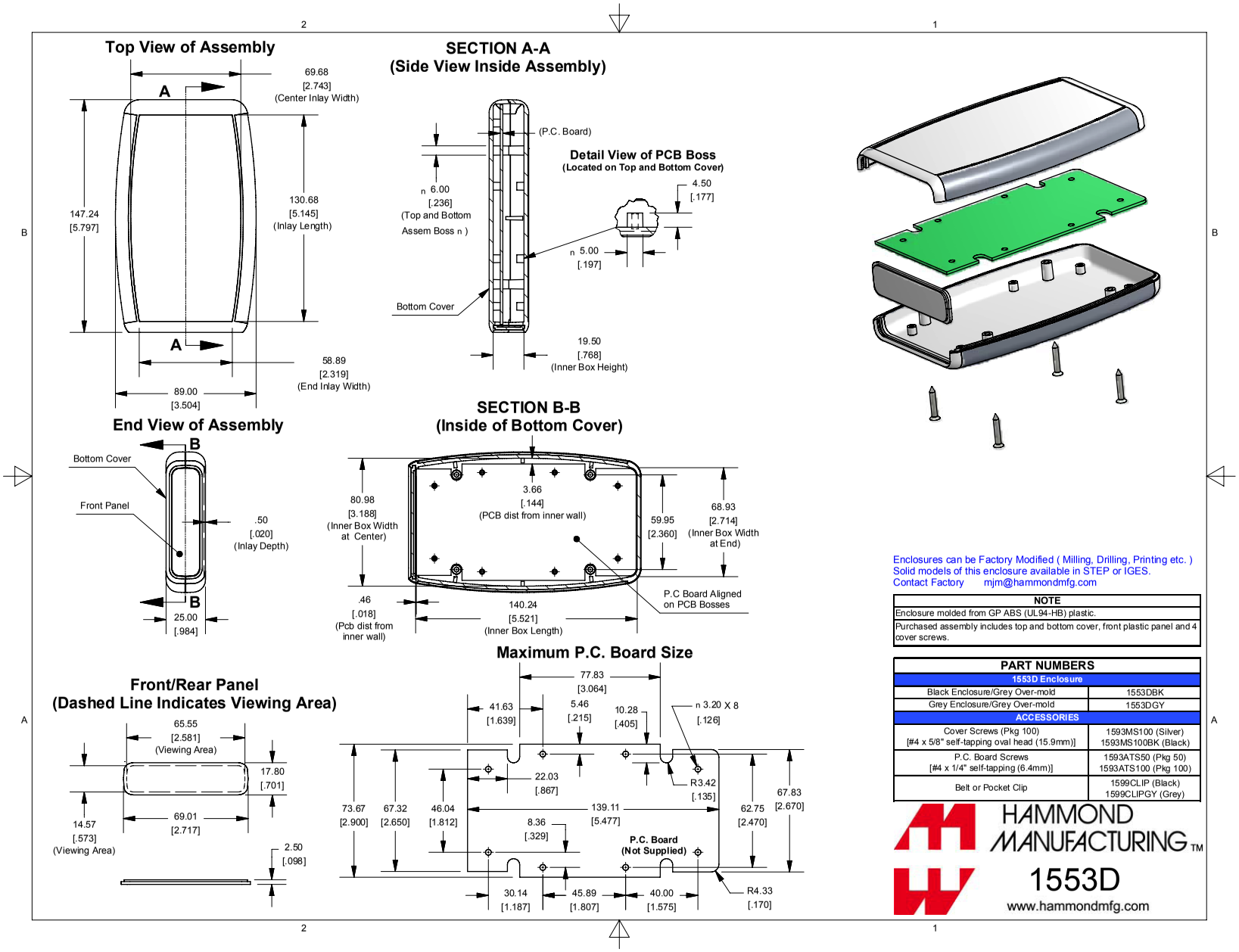


Figure B-5: Hammond Enclosure Datasheet

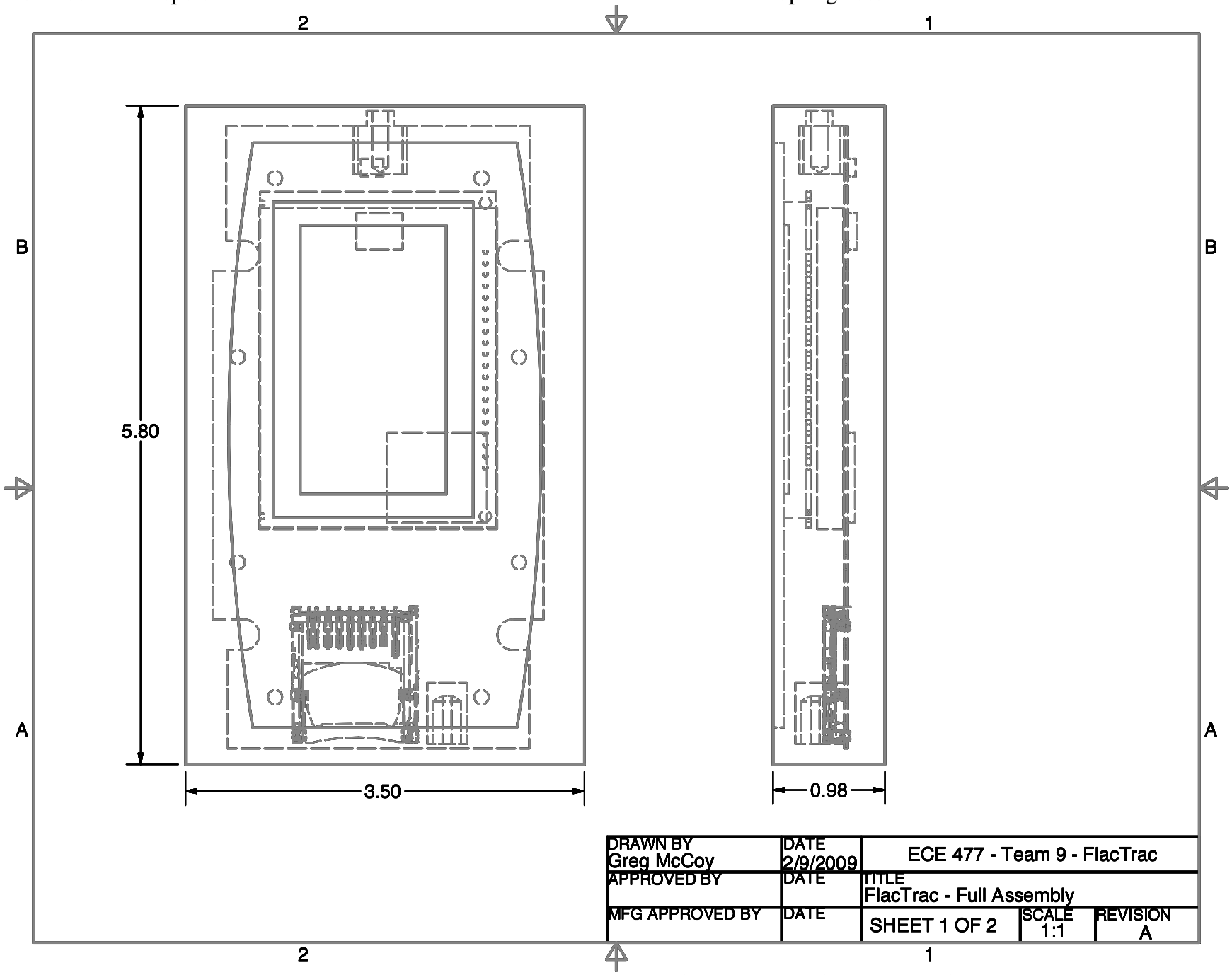


Figure B-6: Enclosure/Internals Clearance

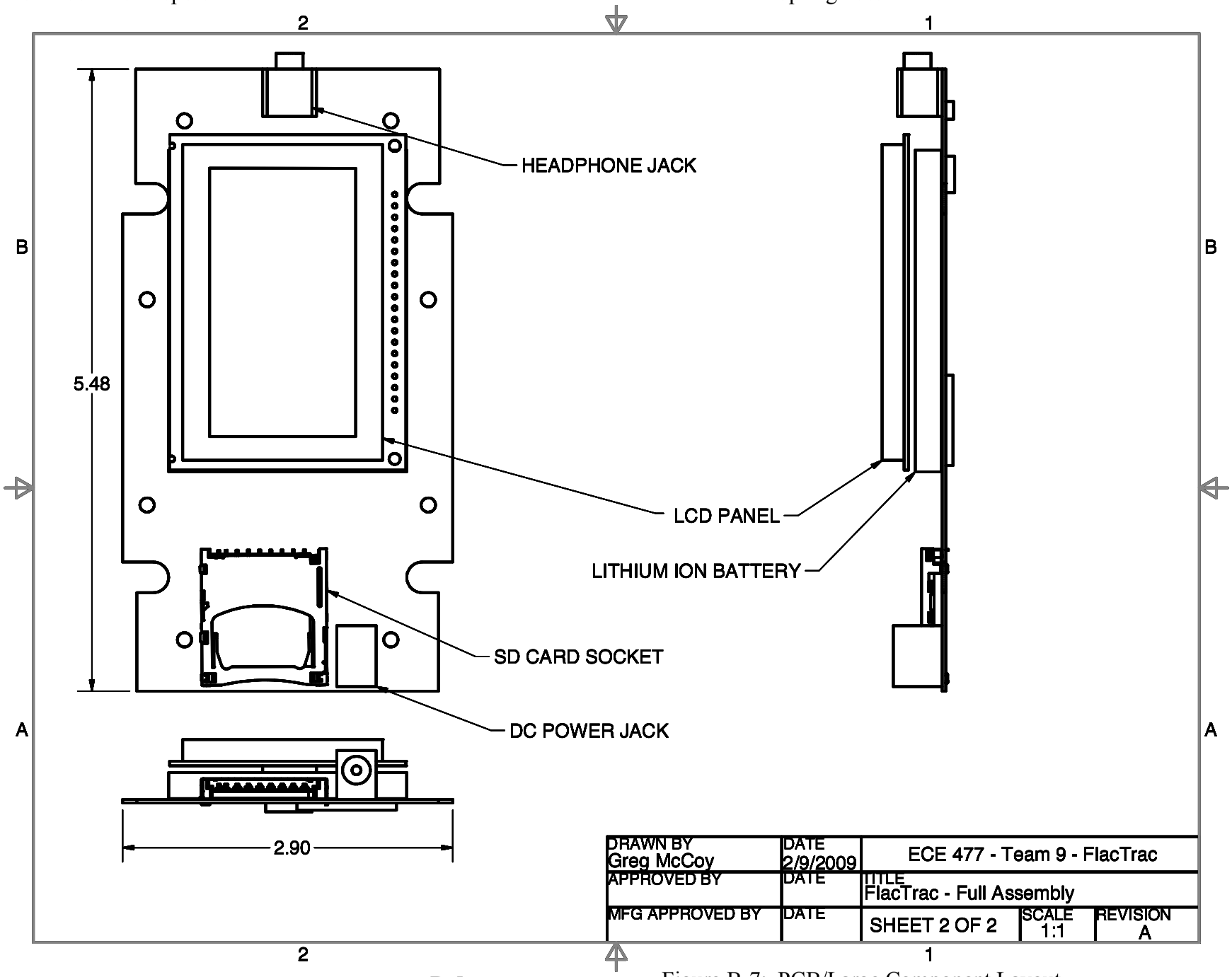


Figure B-7: PCB/Large Component Layout

Appendix C: Schematic

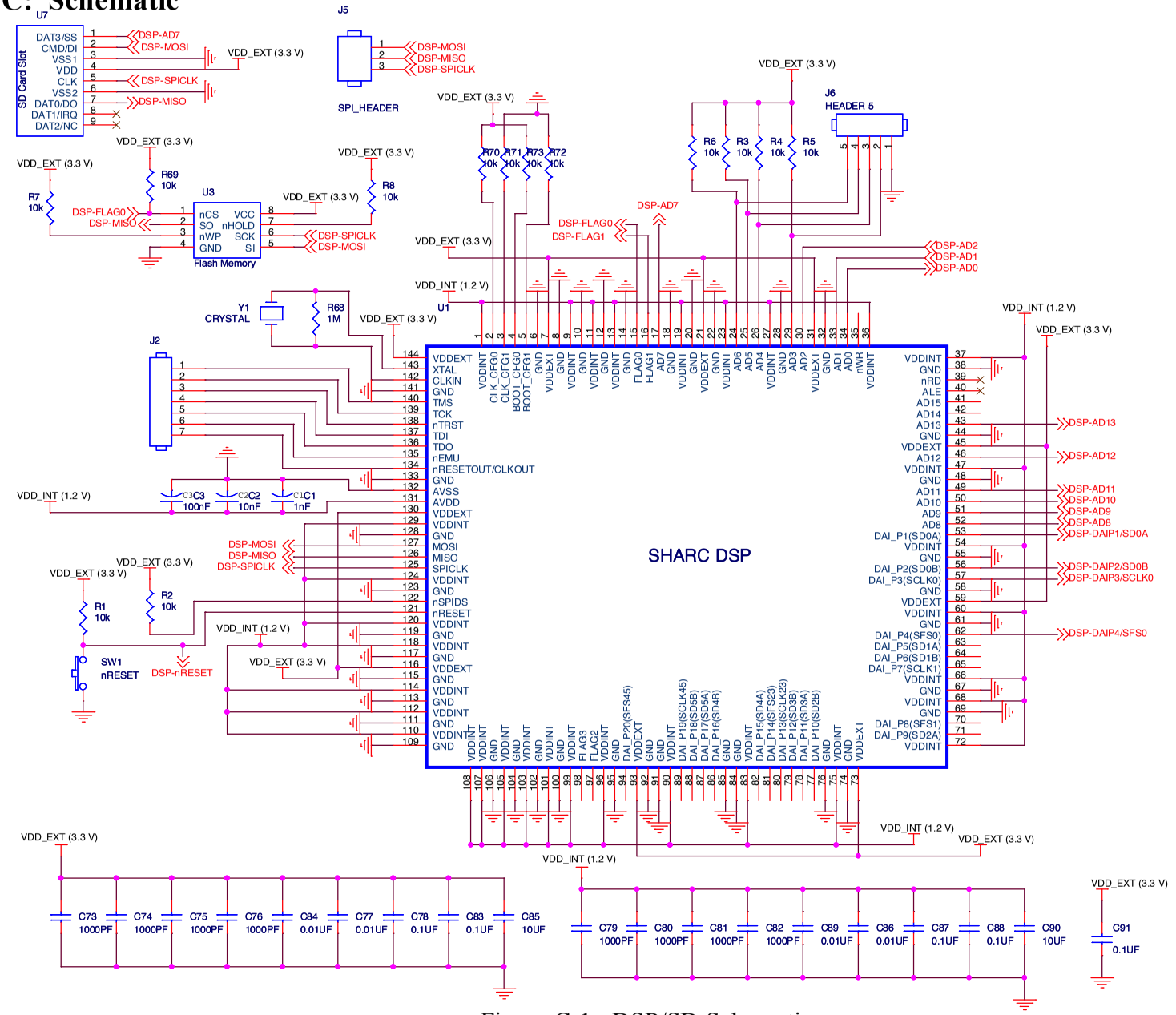


Figure C-1: DSP/SD Schematic

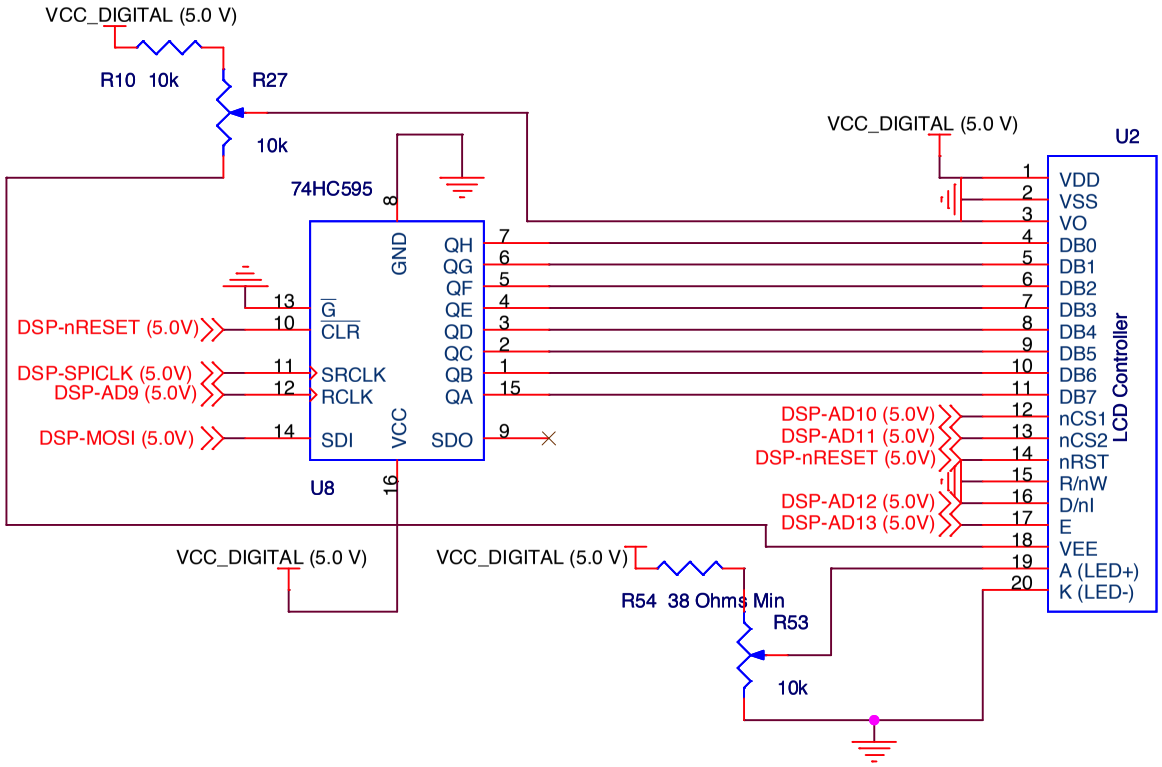


Figure C-2: LCD Schematic

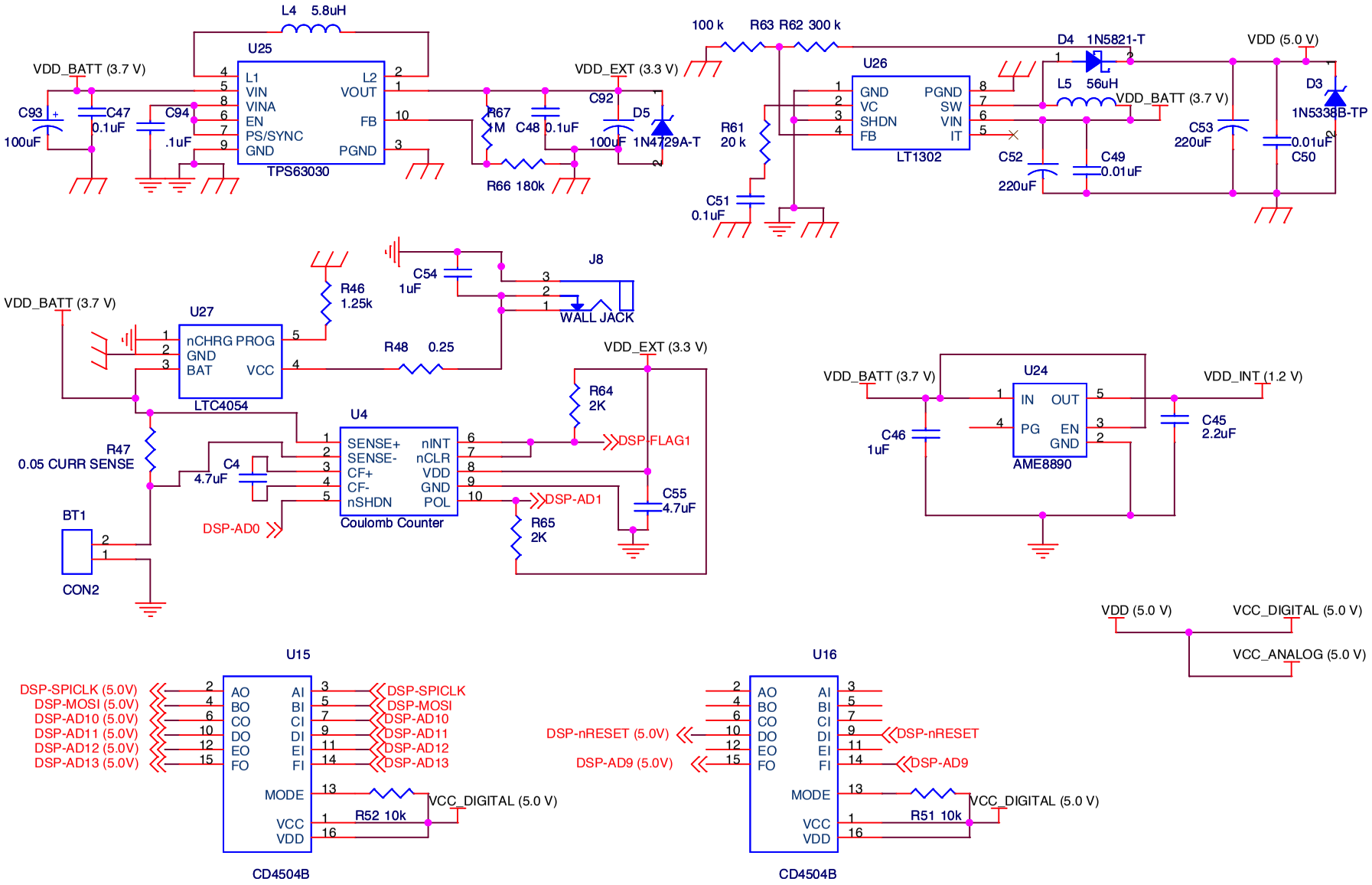


Figure C-3: Power Schematic

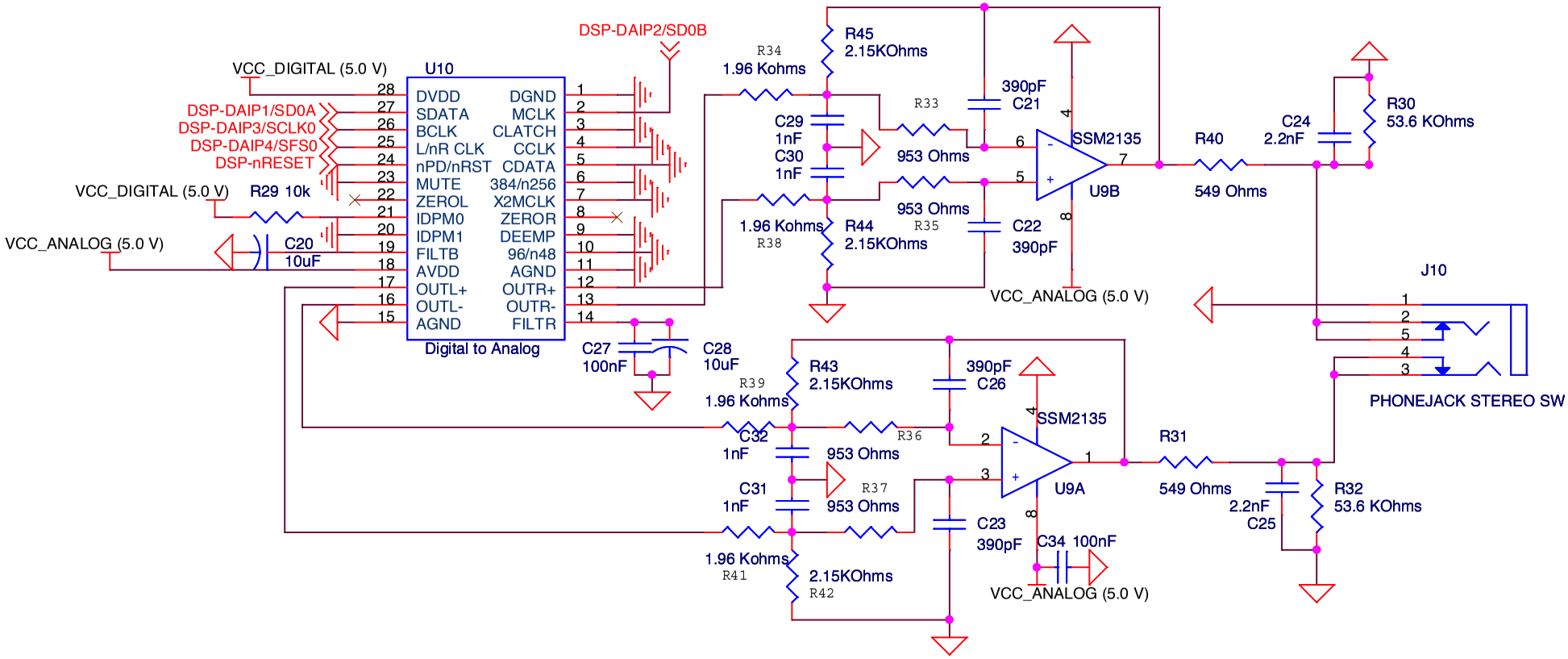


Figure C-4: Audio Schematic

Appendix D: PCB Layout Top and Bottom Copper

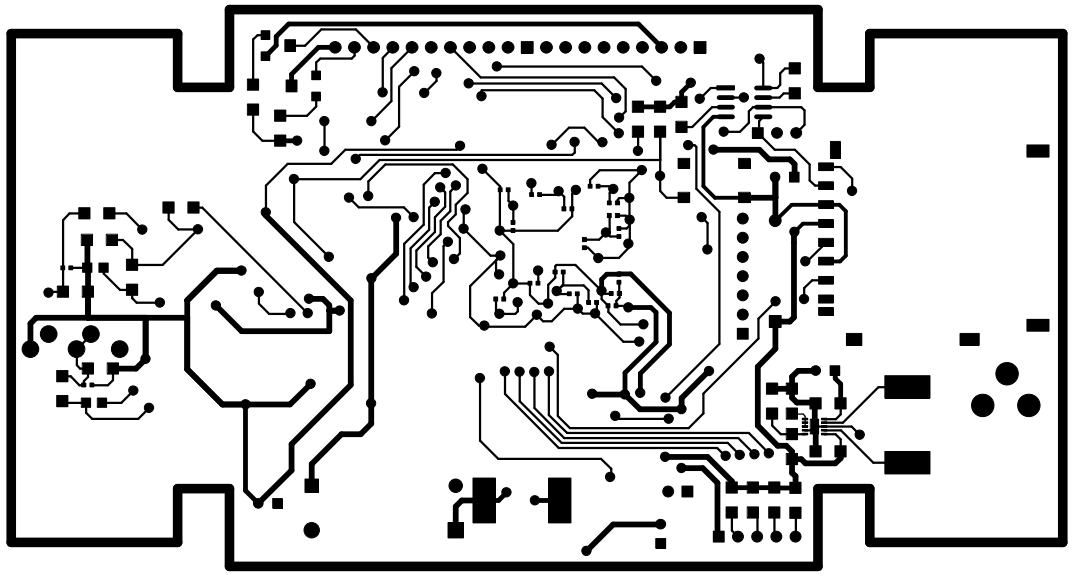


Figure D-1: PCB Layout Top Copper (1:1 scale)

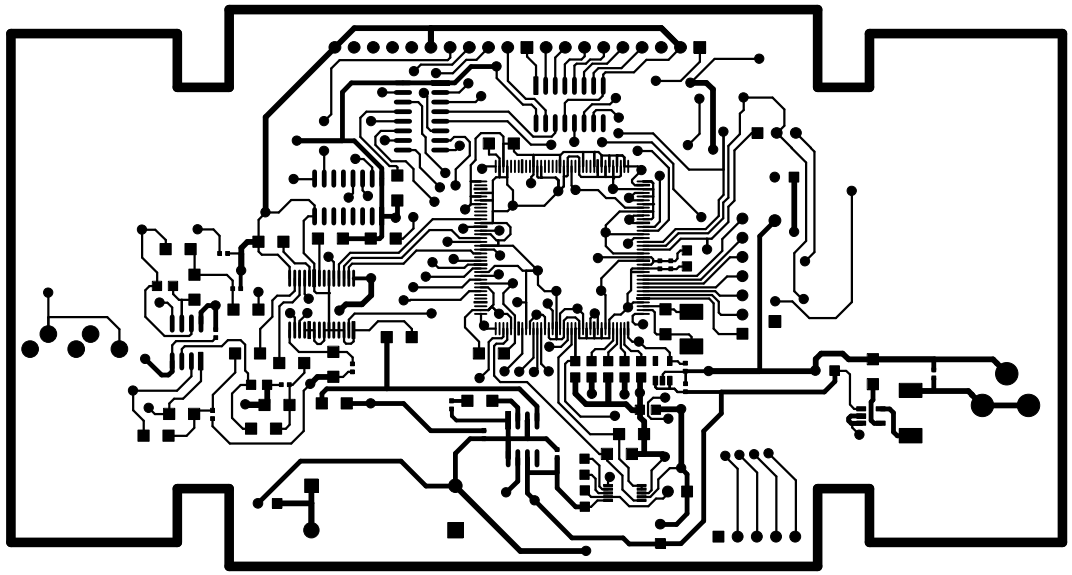


Figure D-2: PCB Layout Bottom Copper (1:1 scale)

Appendix E: Parts List Spreadsheet

Vendor	Manufacturer	Part No.	Description	Category	Package	Qty	Prototype		Production	
							Unit Cost	Total Cost	Unit Cost	Total Cost
Digikey	Analog Devices	ADSP-21262	3rd Gen Low Cost 32-Bit SHARC DSP	Processor	144-LQFP	1	$30.69	$30.69	$25.58	$25.58
Crystalfontz	Crystalfontz	CFAG12864BTMIV	128x64 Graphic LCD w/ Parallel Iface	LCD	PCB	1	$33.30	$33.30	$10.25	$10.25
Mouser	Atmel	AT25F2048N-10SU-2.7	2 Mbit SPI Flash	Processor	8-SOIC	1	$3.45	$3.45	$2.74	$2.74
Digikey	Analog Devices	AD1854JRSZ	Stereo, 96 kHz, Sigma-Delta DAC	Audio Out	28-SSOP	1	$8.80	$8.80	$6.60	$6.60
Digikey	Analog Devices	SSM2135SZ	Dual Audio Operational Amplifier	Audio Out	8-SOIC	1	$5.65	$5.65	$3.01	$3.01
Mouser	TI	SN74HC595DTE4	SPI-Compatible 8 Bit Shift Register	LCD	16-SOIC	1	$0.90	$0.90	$0.38	$0.38
Mouser	TI	CD4504BM	Logic Level Translator	LCD	16-SOIC	2	$0.80	$1.60	$0.24	$0.49
Mouser	Hammond Mfg	1553DGY	Handheld Plastic Enclosure	Packaging	N/A	1	$7.48	$7.48	$4.86	$4.86
Mouser	Bourns	TC33X-2-103E	10K Trimmer Potentiometer	LCD	N/A	2	$0.20	$0.40	$0.07	$0.14
Mouser	KOA Speer	SR732ATTER100F	0.1 1/8 W Resistor	Discrete	0805	1	$0.61	$0.61	$0.16	$0.16
Mouser	KOA Speer	SR733ATTER255F	0.255 1 W Resistor	Discrete	2512	1	$0.89	$0.89	$0.59	$0.59

Vendor	Manufacturer	Part No.	Description	Category	Package	Qty	Prototype		Production	
							Unit Cost	Total Cost	Unit Cost	Total Cost
Mouser	Vishay/Dale	CRCW1206549RFKEA	549 1/4 W Resistor	Discrete	1206	3	$0.10	$0.30	$0.07	$0.21
Mouser	Vishay/Dale	CRCW1206953RFKTA	953 1/4 W Resistor	Discrete	1206	4	$0.10	$0.40	$0.01	$0.03
Mouser	Vishay/Dale	CRCW12061K24FKEA	1.24K 1/4 W Resistor	Discrete	1206	1	$0.10	$0.10	$0.01	$0.01
Mouser	Vishay/Dale	CRCW12061K96FKEA	1.96K 1/4 W Resistor	Discrete	1206	6	$0.10	$0.60	$0.01	$0.04
Mouser	Vishay/Dale	CRCW12062K15FKEA	2.15K 1/4 W Resistor	Discrete	1206	4	$0.10	$0.40	$0.01	$0.03
Mouser	Vishay/Dale	CRCW120653K6FKEA	53.6K 1/4 W Resistor	Discrete	1206	2	$0.10	$0.20	$0.01	$0.01
Mouser	KOA Speer	RK73B2ATTDD103J	10K 1/8 W Resistor	Discrete	0805	5	$0.06	$0.30	$0.00	$0.01
Mouser	Vishay/Dale	CRCW120610K0FKEB	10K 1/4 W Resistor	Discrete	1206	12	$0.10	$1.20	$0.01	$0.08
Mouser	KOA Speer	RK73B2BTTD203J	20K 1/4 W Resistor	Discrete	1206	1	$0.05	$0.05	$0.00	$0.00
Mouser	KOA Speer	RK73H2BTTDD1003F	100K 1/4 W Resistor	Discrete	1206	1	$0.05	$0.05	$0.01	$0.01
Mouser	KOA Speer	RK73H2BTTD1803F	180 K 3/4 W Resistor	Discrete	1206	1	$0.10	$0.10	$0.00	$0.00
Mouser	Vishay/Dale	CRCW1206300KFKEA	300K 1/4 W Resistor	Discrete	1206	1	$0.10	$0.10	$0.01	$0.01
Mouser	Vishay/Dale	CRCW12061M00FKEA	1M 1/4 W Resistor	Discrete	1206	2	$0.10	$0.20	$0.01	$0.01
Mouser	AVX	04025C102JAT2A	1000 pF Capacitor	Discrete	0402	8	$0.41	$3.28	$0.05	$0.42
Mouser	Kemet	C1206C104J5RAC7025	0.1 µF Capacitor	Discrete	1206	3	$0.28	$0.84	$0.04	$0.12
Mouser	AVX	0402ZD104KAT2A	.1 µF Capacitor	Discrete	0402	8	$0.11	$0.88	$0.03	$0.21

Vendor	Manufacturer	Part No.	Description	Category	Package	Qty	Prototype		Production	
							Unit Cost	Total Cost	Unit Cost	Total Cost
Mouser	AVX	0402YC103KAT2A	0.01 µF Capacitor	Discrete	0402	6	$0.12	$0.72	$0.01	$0.08
Mouser	Kemet	C0402C105K8PACTU	1 µF Capacitor	Discrete	0402	2	$0.35	$0.70	$0.09	$0.18
Mouser	AVX	0805YD475KAT2A	4.7 µF Capacitor	Discrete	0805	2	$0.66	$1.32	$0.21	$0.42
Mouser	AVX	1206ZD106KAT2A	10 µF Capacitor	Discrete	1206	4	$1.21	$4.84	$0.15	$0.58
Mouser	AVX	04025C222KAT2A	2.2 nF Capacitor	Discrete	0402	3	$0.28	$0.84	$0.03	$0.10
Mouser	AVX	08051A391JAT2A	390 pF Capacitor	Discrete	0805	4	$0.14	$0.56	$0.05	$0.18
Mouser	AVX	04023C102JAT2A	1 nF Capacitor	Discrete	0402	5	$0.41	$2.05	$0.07	$0.34
Mouser	AVX	08051C103KAT2A	10 nF Capacitor	Discrete	0805	1	$0.11	$0.11	$0.04	$0.04
Mouser	AVX	0402YG104ZAT2A	100 nF Capacitor	Discrete	0402	2	$0.22	$0.44	$0.03	$0.06
Mouser	Nichicon	UVR0J101MDD	100 µF Capacitor	Discrete	Thru-Hole	2	$0.18	$0.36	$0.03	$0.07
Mouser	Nichicon	UPW1A221MED	220 µF Capacitor	Discrete	Thru-Hole	2	$0.24	$0.48	$0.10	$0.20
Mouser	API Delevan	1812-562J	5.8 µH Inductor	Discrete	N/A	1	$0.73	$0.73	$0.46	$0.46
Mouser	API Delevan	4922R-22L	56 µH Inductor	Discrete	N/A	1	$2.76	$2.76	$1.31	$1.31
Mouser	ON Semiconductor	1N5338BG	Zener Diode	Discrete	Axial	1	$0.38	$0.38	$0.16	$0.16
Mouser	Diodes, Inc.	1N5821-T	Schottkey Diode	Discrete	Axial	1	$0.67	$0.67	$0.18	$0.18
Mouser	Diodes, Inc.	1N4729A-T	Zener Diode	Discrete	Axial	1	$0.36	$0.36	$0.06	$0.06
Digikey	Citizen	CS10 12.500MABJ-UT	12.5 MHz Crystal	Discrete	N/A	1	$1.24	$1.24	$0.79	$0.79

Vendor	Manufacturer	Part No.	Description	Category	Package	Qty	Prototype		Production	
							Unit Cost	Total Cost	Unit Cost	Total Cost
Mouser	Kobiconn	163-7620-E	Coaxial DC Power Jack	Power	PCB	1	$0.63	$0.63	$0.15	$0.15
Mouser	Ultralife	UBBP01	Lithium Ion Rechargeable Battery	Power	PCB	1	$14.45	$14.45	$10.58	$10.58
Digikey	Linear Technology	LT1302CS8#PBF	DC/DC Step-Up Converter	Power	8-SOIC	1	$6.38	$6.38	$3.50	$3.50
Mouser	AME	AME8890TEEV	Low Dropout Voltage Regulator 1.2 V 150mA CMOS	Power	SOT-25	1	$0.72	$0.72	$0.20	$0.20
Digikey	Linear Technology	LTC4054LES5-4.2#TRPBF	Lithium Ion Charging IC	Power	TSOT-23-5	1	$3.47	$3.47	$1.82	$1.82
Digikey	Linear Technology	LTC4150CMS#PBF	Coulomb Counter	Power	10-MSOP	1	$3.00	$3.00	$1.70	$1.70
Mouser	TI	TPS63030DSKT	Voltage Regulator	Power		1	$4.00	$4.00	$2.02	$2.02
Digikey	Omron	B3FS-1052	Surface Mount Pushbutton	Processor	PCB	1	$2.09	$2.09	$1.12	$1.12
Mouser	Molex	67840-8001	SD Card Connector	Audio Out	PCB	1	$3.66	$3.66	$2.46	$2.46
Mouser	Kobiconn	161-354W-EX	3.5mm Stereo Audio Jack	Audio Out	PCB	1	$1.32	$1.32	$0.69	$0.69
					Total	124	Total	$161.05	Total	$85.46

Table E-1: Bill of Materials

ID	P/N	Description	Category	Footprint	Value
BT1	UBBP01	Lithium Ion Rechargeable Battery	Power - Battery		

ID	P/N	Description	Category	Footprint	Value
C1	04023C102JAT2A	1 nF Capacitor	Power - SHARC Decoupling	SM/C_0402	1nF
C2	08051C103KAT2A	10 nF Capacitor	Power - SHARC Decoupling	SM/C_0805	10nF
C3	0402YG104ZAT2A	100 nF Capacitor	Power - SHARC Decoupling	SM/C_0402	100nF
C4	0805YD475KAT2A	4.7 µF Capacitor	Power - Battery	SM/C_0805	4.7µF
C20	1206ZD106KAT2A	10 µF Capacitor	Audio Out	SM/C_1206	10µF
C21	08051A391JAT2A	390 pF Capacitor	Audio Out	SM/C_0805	390pF
C22	08051A391JAT2A	390 pF Capacitor	Audio Out	SM/C_0805	390pF
C23	08051A391JAT2A	390 pF Capacitor	Audio Out	SM/C_0805	390pF
C24	04025C222KAT2A	2.2 nF Capacitor	Audio Out	SM/C_0402	2.2nF
C25	04025C222KAT2A	2.2 nF Capacitor	Audio Out	SM/C_0402	2.2nF
C26	08051A391JAT2A	390 pF Capacitor	Audio Out	SM/C_0805	390pF
C27	0402YG104ZAT2A	100 nF Capacitor	Audio Out	SM/C_0402	100nF
C28	1206ZD106KAT2A	10 µF Capacitor	Audio Out	SM/C_1206	10µF
C29	04023C102JAT2A	1 nF Capacitor	Audio Out	SM/C_0402	1nF
C30	04023C102JAT2A	1 nF Capacitor	Audio Out	SM/C_0402	1nF
C31	04023C102JAT2A	1 nF Capacitor	Audio Out	SM/C_0402	1nF
C32	04023C102JAT2A	1 nF Capacitor	Audio Out	SM/C_0402	1nF
C33	0402ZD104KAT2A	.1 µF Capacitor	Audio Out		
C34	0402ZD104KAT2A	.1 µF Capacitor	Audio Out	SM/C_0402	100nF
C45	04025C222KAT2A	2.2 nF Capacitor	Power - AME8890 Circuit	SM/C_0402	2.2µF
C46	C0402C105K8PACTU	1 µF Capacitor	Power - AME8890 Circuit	SM/C_0402	1µF
C47	C1206C104J5RAC7025	0.1 µF Capacitor	Power - TPS63030	SM/C_1206	0.1µF
C48	C1206C104J5RAC7025	0.1 µF Capacitor	Power - TPS63030	SM/C_1206	0.1µF
C49	0402YC103KAT2A	0.01 µF Capacitor	Power - LT1302 Circuit	SM/C_0402	0.01µF
C50	0402YC103KAT2A	0.01 µF Capacitor	Power - LT1302 Circuit	SM/C_0402	0.01µF

ID	P/N	Description	Category	Footprint	Value
C51	0402ZD104KAT2A	.1 µF Capacitor	Power - LT1302 Circuit	SM/C_0402	0.1µF
C52	UPW1A221MED	220 µF Capacitor	Power - LT1302 Circuit	CPCYL1/D.325/LS.125/.034	220µF
C53	UPW1A221MED	220 µF Capacitor	Power - LT1302 Circuit	CPCYL1/D.325/LS.125/.034	220µF
C54	C0402C105K8PACTU	1 µF Capacitor	Power - Battery	SM/C_0402	1µF
C55	0805YD475KAT2A	4.7 µF Capacitor	Power - Battery	SM/C_0805	4.7µF
C73	04025C102JAT2A	1000 pF Capacitor	Power - SHARC Decoupling	SM/C_0402	1000PF
C74	04025C102JAT2A	1000 pF Capacitor	Power - SHARC Decoupling	SM/C_0402	1000PF
C75	04025C102JAT2A	1000 pF Capacitor	Power - SHARC Decoupling	SM/C_0402	1000PF
C76	04025C102JAT2A	1000 pF Capacitor	Power - SHARC Decoupling	SM/C_0402	1000PF
C77	0402YC103KAT2A	0.01 µF Capacitor	Power - SHARC Decoupling	SM/C_0402	0.01µF
C78	0402ZD104KAT2A	.1 µF Capacitor	Power - SHARC Decoupling	SM/C_0402	0.1µF
C79	04025C102JAT2A	1000 pF Capacitor	Power - SHARC Decoupling	SM/C_0402	1000PF
C80	04025C102JAT2A	1000 pF Capacitor	Power - SHARC Decoupling	SM/C_0402	1000PF
C81	04025C102JAT2A	1000 pF Capacitor	Power - SHARC Decoupling	SM/C_0402	1000PF
C82	04025C102JAT2A	1000 pF Capacitor	Power - SHARC Decoupling	SM/C_0402	1000PF
C83	0402ZD104KAT2A	.1 µF Capacitor	Power - SHARC Decoupling	SM/C_0402	0.1µF
C84	0402YC103KAT2A	0.01 µF Capacitor	Power - SHARC Decoupling	SM/C_0402	0.01µF
C85	1206ZD106KAT2A	10 µF Capacitor	Power - SHARC Decoupling	SM/C_1206	10µF
C86	0402YC103KAT2A	0.01 µF Capacitor	Power - SHARC Decoupling	SM/C_0402	0.01µF

ID	P/N	Description	Category	Footprint	Value
C87	0402ZD104KAT2A	.1 µF Capacitor	Power - SHARC Decoupling	SM/C_0402	0.1µF
C88	0402ZD104KAT2A	.1 µF Capacitor	Power - SHARC Decoupling	SM/C_0402	0.1µF
C89	0402YC103KAT2A	0.01 µF Capacitor	Power - SHARC Decoupling	SM/C_0402	0.01µF
C90	1206ZD106KAT2A	10 µF Capacitor	Power - SHARC Decoupling	SM/C_1206	10µF
C91	0402ZD104KAT2A	.1 µF Capacitor	Power - SHARC Decoupling	SM/C_0402	0.1µF
C92	UVR0J101MDD	100 µF Capacitor	Power - TPS63030	CPCYL1/D.200/LS.100/.031	100µF
C93	UVR0J101MDD	100 µF Capacitor	Power - TPS63030	CPCYL1/D.200/LS.100/.031	100µF
C94	C1206C104J5RAC7025	0.1 µF Capacitor	Power - TPS63030	SM/C_1206	0.1µF
D3	1N5338BG	Zener Diode	Power - LT1302 Circuit	DAX1/1N_5333B-5388B	1N5338B-TP
D4	1N5821-T	Schottkey Diode	Power - LT1302 Circuit	DAX1/1N_5820-5821	1N5821-T
D5	1N4729A-T	Zener Diode	Power - TPS63030	DAX1/1N_4728A-4764A	1N4729A-T
J2	To Do		Headers	BLKCON.100/VH/TM1SQ/W.100/7	HEADER 7
J3	161-354W-EX	3.5mm Stereo Audio Jack	Audio Out		
J5	To Do		Headers		
J7	To Do		Power - Battery		
J8	163-7620-E	Coaxial DC Power Jack	Power - Battery		
L4	1812-562J	5.8 µH Inductor	Power - TPS63030	SM/L_CDRH125	5.8uH
L5	4922R-22L	56 µH Inductor	Power - LT1302 Circuit	SM/L_CDRH125	56uH
M1	1553DGY	Handheld Plastic Enclosure	Mechanical		
R1	CRCW120610K0FKEB	10K 1/4 W Resistor	SHARC/SHARC IF	SM/R_1206	10k
R2	CRCW120610K0FKEB	10K 1/4 W Resistor	SHARC/SHARC IF	SM/R_1206	10k

ID	P/N	Description	Category	Footprint	Value
R3	CRCW120610K0FKEB	10K 1/4 W Resistor	SHARC/SHARC IF	SM/R_1206	10k
R4	CRCW120610K0FKEB	10K 1/4 W Resistor	Buttons	SM/R_1206	10k
R5	CRCW120610K0FKEB	10K 1/4 W Resistor	Buttons	SM/R_1206	10k
R6	CRCW120610K0FKEB	10K 1/4 W Resistor	Buttons	SM/R_1206	10k
R7	CRCW120610K0FKEB	10K 1/4 W Resistor	Flash Memory	SM/R_1206	10k
R8	CRCW120610K0FKEB	10K 1/4 W Resistor	Buttons	SM/R_1206	10k
R10	CRCW120610K0FKEB	10K 1/4 W Resistor	LCD	SM/R_1206	10k
R27	TC33X-2-103E	10K Trimmer Potentiometer	LCD	TRIMMER10K	10k
R29	CRCW120610K0FKEB	10K 1/4 W Resistor	Audio Out	SM/R_1206	10k
R30	CRCW120653K6FKEA	53.6K 1/4 W Resistor	Audio Out	SM/R_1206	53.6 KOhms
R31	CRCW1206549RFKEA	549 1/4 W Resistor	Audio Out	SM/R_1206	549 Ohms
R32	CRCW120653K6FKEA	53.6K 1/4 W Resistor	Audio Out	SM/R_1206	53.6 KOhms
R33	CRCW1206953RFKTA	953 1/4 W Resistor	Audio Out	SM/R_1206	953 Ohms
R34	CRCW12061K96FKEA	1.96K 1/4 W Resistor	Audio Out	SM/R_1206	1.96 Kohms
R35	CRCW1206953RFKTA	953 1/4 W Resistor	Audio Out	SM/R_1206	953 Ohms
R36	CRCW1206953RFKTA	953 1/4 W Resistor	Audio Out	SM/R_1206	953 Ohms
R37	CRCW1206953RFKTA	953 1/4 W Resistor	Audio Out	SM/R_1206	953 Ohms
R38	CRCW12061K96FKEA	1.96K 1/4 W Resistor	Audio Out	SM/R_1206	1.96 Kohms
R39	CRCW12061K96FKEA	1.96K 1/4 W Resistor	Audio Out	SM/R_1206	1.96 Kohms
R40	CRCW1206549RFKEA	549 1/4 W Resistor	Audio Out	SM/R_1206	549 Ohms
R41	CRCW12061K96FKEA	1.96K 1/4 W Resistor	Audio Out	SM/R_1206	1.96 Kohms
R42	CRCW12062K15FKEA	2.15K 1/4 W Resistor	Audio Out	SM/R_1206	2.15KOhms
R43	CRCW12062K15FKEA	2.15K 1/4 W Resistor	Audio Out	SM/R_1206	2.15KOhms
R44	CRCW12062K15FKEA	2.15K 1/4 W Resistor	Audio Out	SM/R_1206	2.15KOhms
R45	CRCW12062K15FKEA	2.15K 1/4 W Resistor	Audio Out	SM/R_1206	2.15KOhms
R46	CRCW12061K24FKEA	1.24K 1/4 W Resistor	Power - Battery	SM/R_1206	1.25k
R47	SR732ATTER100F	0.1 1/8 W Resistor	Power - Battery	SM/R_0805	0.05 CURR SENSE
R48	SR733ATTER255F	0.255 1 W Resistor	Power - Battery	SM/R_2512	0.25
R51	CRCW120610K0FKEB	10K 1/4 W Resistor	SHARC/SHARC IF	SM/R_1206	10k
R52	CRCW120610K0FKEB	10K 1/4 W Resistor	SHARC/SHARC IF	SM/R_1206	10k
R53	TC33X-2-103E	10K Trimmer Potentiometer	LCD	TRIMMER10K	10k

ID	P/N	Description	Category	Footprint	Value
R54	CRCW1206549RFKEA	549 1/4 W Resistor	LCD	SM/R_1206	38 Ohms Min
R61	RK73B2BTTD203J	20K 1/4 W Resistor	Power - LT1302 Circuit	SM/R_1206	20 k
R62	CRCW1206300KFKEA	300K 1/4 W Resistor	Power - LT1302 Circuit	SM/R_1206	300 k
R63	RK73H2BTTDD1003F	100K 1/4 W Resistor	Power - LT1302 Circuit	SM/R_1206	100 k
R64	CRCW12061K96FKEA	1.96K 1/4 W Resistor	Power - Battery	SM/R_1206	2K
R65	CRCW12061K96FKEA	1.96K 1/4 W Resistor	Power - Battery	SM/R_1206	2K
R66	RK73H2BTTD1803F	180 K 3/4 W Resistor	Power - TPS63030	SM/R_1206	180k
R67	CRCW12061M00FKEA	1M 1/4 W Resistor	Power - TPS63030	SM/R_1206	1M
R68	CRCW12061M00FKEA	1M 1/4 W Resistor	SHARC/SHARC IF	SM/R_1206	1M
R69	RK73B2ATTDD103J	10K 1/8 W Resistor	SHARC/SHARC IF	SM/R_0805	10k
R70	RK73B2ATTDD103J	10K 1/8 W Resistor	SHARC/SHARC IF	SM/R_0805	10k
R71	RK73B2ATTDD103J	10K 1/8 W Resistor	SHARC/SHARC IF	SM/R_0805	10k
R72	RK73B2ATTDD103J	10K 1/8 W Resistor	SHARC/SHARC IF	SM/R_0805	10k
R73	RK73B2ATTDD103J	10K 1/8 W Resistor	SHARC/SHARC IF	SM/R_0805	10k
SW1	B3FS-1052	Surface Mount Pushbutton	SHARC/SHARC IF		
U1	ADSP-21262	3rd Gen Low Cost 32-Bit SHARC DSP	SHARC/SHARC IF		
U2	CFAG12864BTMIV	128x64 Graphic LCD w/ Parallel Iface	LCD		
U3	AT25F2048N-10SU-2.7	2 Mbit SPI Flash	Flash Memory		
U4	LTC4150CMS#PBF	Coulomb Counter	Power - Battery		
U7	67840-8001	SD Card Connector	SD Card		
U8	SN74HC595DTE4	SPI-Compatible 8 Bit Shift Register	LCD		
U9	SSM2135SZ	Dual Audio Operational Amplifier	Audio Out		
U10	AD1854JRSZ	Stereo, 96 kHz, Sigma-Delta DAC	Audio Out		
U15	CD4504BM	Logic Level Translator	SHARC/SHARC IF		

ID	P/N	Description	Category	Footprint	Value
U16	CD4504BM	Logic Level Translator	SHARC/SHARC IF		
U24	AME8890TEEV	Low Dropout Voltage Regulator 1.2 V 150mA CMOS	Power - AME8890 Circuit		
U25	TPS63030DSKT	Voltage Regulator	Power - TPS63030		
U26	LT1302CS8#PBF	DC/DC Step-Up Converter	Power - LT1302 Circuit		
U27	LTC4054LES5-4.2#TRPBF	Lithium Ion Charging IC	Power - Battery		
X1	CS10 12.500MABJ-UT	12.5 MHz Crystal	SHARC/SHARC IF		

Table E-2: Parts Specification

Appendix F: Software Listing

```
/*  Title:    FlacTrac
 *  Version:   1.0
 *  Filename:  FlacTrac.c
 *  Authors:   Brett Mravec, Isaac Jones, Greg McCoy
 *  Purpose:   Main software routines for device
 *  Date:      7 Feb 2009
 */

#include "FlacTrac.h"
#include "flac.h"
#include "raw.h"
#include "lcd.h"
#include "audio.h"
#include "sd.h"
#include "bit-reader.h"
#include "util.h"
#include "spi.h"

#include <sysreg.h>
#include <signal.h>
#include <21262.h>

typedef enum _IState IState;
enum _IState {
    STATE_BROWSING,
    STATE_PLAYING,
    STATE_PAUSED,
    STATE_ERROR
};

static char count;
static char tenth_sec, eight_ms;
static char buttons;
static char scroll_line;

static IState state;

static char (*decoder_read_frame) (void);

void
main ()
{
    int i, j, len;
    int tmp;

    *pSPICTL = 0;
    *pSPIFLG = 0;
    *pSPIBAUD = 0;
    *pSPCTL0 = 0;
    *pSYSCTL |= 0x100000;
    *pCSP0A = 0;

    lcd_init ();
```

```
    lcd_clear ();
    lcd_printf ("SD Init...     ", 0, 0);
    lcd_update ();
    sd_init ();
    lcd_printf ("Done\n", 0, 0);
    lcd_printf ("Fat Init...    ", 0, 0);
    lcd_update ();
    fat_init ();
    lcd_printf ("Done\n", 0, 0);
    lcd_update ();

    tmp = 0;
    count = 0;
    tenth_sec = 0;
    eight_ms = 0;
    buttons = 0x0f;
    scroll_line = 0;

    FileEntry *files;
    files = fat_ls ();

    state = STATE_BROWSING;
    lcd_print_directories (files, scroll_line);
    lcd_update ();

    i = 0;

    // Setup timer interrupt
    interrupt (SIG_TMZ, timer_isr);
    timer_set (200000, 200000);
    timer_on ();

    while (2) {
        if (state == STATE_PLAYING) {
            if (decoder_read_frame ()) {
                state = STATE_BROWSING;
                fat_close ();
                audio_clear ();
            }
        }

        if (tenth_sec) {
            tenth_sec = 0;
            if (state == STATE_PLAYING) {
                lcd_clear ();
                lcd_print_playback(tmp/10);
                lcd_update ();

                tmp = (tmp + 1) % 1000;
            } else if (state == STATE_BROWSING) {
                lcd_clear ();
                lcd_print_directories (files, scroll_line);
                lcd_update ();
            }
        }

        if (eight_ms) {
            eight_ms = 0;
```

```
char nb = flactrac_poll_buttons ();

if (nb & BTN1 && !(buttons & BTN1)) {
    if (state == STATE_BROWSING) {
        if (scroll_line == 7) {
            fat_ls_down ();
        } else {
            if (files[scroll_line+1].type != '\0')
                scroll_line++;
        }
    } else if (state == STATE_ERROR) {
        state = STATE_BROWSING;
    }
}

if (nb & BTN2 && !(buttons & BTN2)) {
    if (state == STATE_BROWSING) {
        if (scroll_line == 0) {
            fat_ls_up ();
        } else {
            scroll_line--;
        }
    } else if (state == STATE_ERROR) {
        state = STATE_BROWSING;
    }
}

if (nb & BTN3 && !(buttons & BTN3)) {
    if (state == STATE_BROWSING) {
        if (ends_with (files[scroll_line].name, "raw")) {
            fat_open (&files[scroll_line]);
            if (!raw_init ()) {
                state = STATE_PLAYING;
                decoder_read_frame = &raw_read_frame;
                tmp = 0;
            } else {
                fat_close ();

                state = STATE_ERROR;
                lcd_clear ();
                lcd_printf ("Invalid raw file!\n", 0, 0);
                lcd_printf ("Press any key to\ncontinue...\n", 0, 0);
                lcd_update ();
            }
        } else if (ends_with (files[scroll_line].name, "flac")) {
            fat_open (&files[scroll_line]);
            int ret = flac_init ();
            if (!ret) {
                state = STATE_PLAYING;
                decoder_read_frame = &flac_read_frame;
                tmp = 0;
            } else if (ret == -1) {
                fat_close ();

                state = STATE_ERROR;
                lcd_clear ();
                lcd_printf ("Invalid flac file!\n", 0, 0);
```

```
                            lcd_printf ("Press any key to\ncontinue...\n", 0, 0);
                            lcd_update ();
                        } else if (ret == -2) {
                            fat_close ();

                            state = STATE_ERROR;
                            lcd_clear ();
                            lcd_printf ("File read error!\n", 0, 0);
                            lcd_printf ("Press any key to\ncontinue...\n", 0, 0);
                            lcd_update ();
                        }
                    } else {
                        state = STATE_ERROR;
                        lcd_clear ();
                        lcd_printf ("Invalid audio file!\n", 0, 0);
                        lcd_printf ("Press any key to\ncontinue...\n", 0, 0);
                        lcd_update ();
                    }
                } else if (state == STATE_PLAYING) {
                    state = STATE_PAUSED;
                } else if (state == STATE_PAUSED) {
                    state = STATE_PLAYING;
                } else if (state == STATE_ERROR) {
                    state = STATE_BROWSING;
                }
            }

            if (nb & BTN4 && !(buttons & BTN4)) {
                if (state == STATE_BROWSING) {

                } else if (state == STATE_ERROR) {
                    state = STATE_BROWSING;
                } else {
                    fat_close ();
                    audio_clear ();
                    state = STATE_BROWSING;
                }
            }

            buttons = nb;
        }
    }
}

char
flactrac_poll_buttons (void)
{
    char mask = 0;

    if (sysreg_bit_tst (sysreg_FLAGS, FLG11)) {
        mask |= BTN2;
    }

    if (sysreg_bit_tst (sysreg_FLAGS, FLG12)) {
        mask |= BTN4;
    }

    if (sysreg_bit_tst (sysreg_FLAGS, FLG13)) {
```

```
        mask |= BTN1;
    }

    if (sysreg_bit_tst (sysreg_FLAGS, FLG14)) {
        mask |= BTN3;
    }

    return mask;
}

void
timer_isr (int sig_int)
{
    count++;

    if (count == 7) {
        eight_ms = 1;
    }

    if (count == 100) {
        count = 0;
        tenth_sec = 1;
    }
}

/*  Title:     FlacTrac
 *  Version:   1.0
 *  Filename:  FlacTrac.h
 *  Authors:   Brett Mravec, Isaac Jones, Greg McCoy
 *  Purpose:   Main software routines for device
 *  Date:      7 Feb 2009
 */

#ifndef __FLACTRAC_H__
#define __FLACTRAC_H__

#define EVAL_BOARD
#define DEBUG
#define NULL 0;

#include <Cdef21262.h>
#include <def21262.h>
#include <sru21262.h>
#include <signal.h>
#include <sysreg.h>

//#include "sd.h"
//#include "fat.h"

#ifdef DEBUG
//#include <stdio.h>
#endif

char flactrac_poll_buttons (void);
void timer_isr (int sig_int);

#define BTN1    0x01
#define BTN2    0x02
```

```
#define BTN3    0x04
#define BTN4    0x08

#endif /* __FLACTRAC_H__ */
/* Title:  Audio Interface Header
 * Version:  0.1
 * Filename:  Audio.h
 * Authors:  Brett Mravec, Isaac Jones
 * Purpose:  Header file for Audio interface function.
 * Date:  19 Apr 2009
 */

#ifndef __AUDIO_H__
#define __AUDIO_H__

void audio_init (int total_file_samples);
void audio_sport_isr (int sig_int);
void audio_buffer_set_full (void);
short *audio_get_buffer (void);
int audio_get_progress (void);
void audio_clear (void);
void audio_get_samples (short *samples, int num_samples);

#endif
/*  Title:     Bit Reader
 *  Version:   0.5
 *  Filename:  bit-reader.h
 *  Authors:   Brett Mravec
 *  Purpose:   Provides bit data stream
 *  Date:      10 April 2009
 */

#ifndef __BIT_READER_H__
#define __BIT_READER_H__

typedef struct _BitReader BitReader;
struct _BitReader {
    char bit_pos;
    char b;
};

void bit_reader_init (BitReader *br);
int bit_reader_get (BitReader *br, char bits);
int bit_reader_get_s (BitReader *br, char bits);
int bit_reader_get_unary (BitReader *br);
long bit_reader_get_utf8 (BitReader *br);
int bit_reader_get_rice (BitReader *br, int M);

int bit_reader_align (BitReader *br);
long bit_reader_skip (BitReader *br, long bits);

#endif /* __BIT_READER_H__ */
/*  Title:     FAT Filesystem
 *  Version:   1.0
 *  Filename:  fat.h
 *  Authors:   Brett Mravec
 *  Purpose:   FAT filesystem routines for SD card
 *  Date:      28 Mar 2009
```

```
 */

#ifndef __FAT_H__
#define __FAT_H__

#define TYPE_FILE 1
#define TYPE_DIR 2

#define FAT_NUM_ENTRIES 8

typedef struct _FileEntry FileEntry;
struct _FileEntry {
    char name[256];
    char type;
    int cluster;
    int size;
};

void fat_init (void);
void fat_open (FileEntry *fptr);
void fat_close (void);
int fat_skip (int len);
int fat_read (char *buff, int len);
int fat_getc (void);
char fat_eof (void);

FileEntry *fat_ls (void);
void fat_ls_down (void);
void fat_ls_up (void);

void fat_cd (FileEntry *fptr);
FileEntry *fat_pwd (void);

#endif /* __FAT_H__ */
/*  Title:     Flac decoder
 *  Version:   0.5
 *  Filename:  flac.h
 *  Authors:   Brett Mravec
 *  Purpose:   State based flac decoder
 *  Date:      10 April 2009
 */

#ifndef __FLAC_H__
#define __FLAC_H__

char flac_init (void);
char flac_read_frame (void);

char *flac_get_title (void);
char *flac_get_album (void);
char *flac_get_artist (void);

char flac_decode_progress (void);

#endif /* __FLAC_H__ */
/* lcd.h
 *
 * Isaac Jones & Greg McCoy
```

```
 *
 */

#ifndef __LCD_H
#define __LCD_H

#include "fat.h"

void lcd_send_cmd(int type, int columns, char byte);
void lcd_ctrl(char cmd, char value);
void lcd_init(void);
void lcd_update(void);
void lcd_clear(void);
void lcd_printf(char *, char inverted, char wrap);
void lcd_print_directories(FileEntry *files, char active_line);
void lcd_print_playback (int progress);

#define ADDRESS_SETUP_TIME 400  //Time, in ns, between changing LCD control
                                //bits and asserting E to latch them in.
#define DATA_SETUP_TIME 400   //Time, in ns, between changing LCD data bits
                                //and de-asserting E to latch them in.
#define DATA_HOLD_TIME 50     //Time, in ns, between de-asserting E and
                                //changing the data bits to the next value.
#define E_HIGH_BEFORE_DATA 500  //Time, in ns, between asserting E and
                                //changing LCD data bits.
#define E_LOW_BEFORE_ADDRESS 500//Time, in ns, between de-asserting E from the
                                //previous command and changing the address
                                //for the nex commands.
#define BRIEF_DELAY 1000      //Short delay to ensure that the LCD can accept
                                //a new set of data
#define NUMBER_OF_MEMORY_PAGES 8//Number of pages the LCD partitions its data into.
#define BYTES_PER_PAGE 64       //Number of bytes per page in the LCD partitions.

#define LCD_WIDTH  128          //Width of the LCD in pixels.
#define LCD_HEIGHT  64          //Height of the LCD in pixels.
#define CHAR_WIDTH   6          //Width of a character in pixels.
#define CHAR_HEIGHT  8          //Height of a character in pixels.

#define LCD_BAUD 0x2E

/* Mapping of functional pins to DSP output pins */
#define E    0x01               // DAI_PB01_I   E
#define CS1  0x02               // DAI_PB02_I   CS1
#define CS2  0x03               // DAI_PB03_I   CS2
#define RW   0x04               // DAI_PB04_I   R/W
#define DI   0x05               // DAI_PB05_I   R/S D/I
#define SR   0x06               // DAI_PB06_I   74HC595 OE

/* Mapping of LCD functions to arbitrary, but distinct, values for
        code consistency. */
#define WRITE_DATA 1
#define SET_Y 2
#define SET_X 3
#define SET_Z 4
#define SET_POWER 5
#define BOTH_COLUMNS 6

const unsigned int font[][2];
```

```
#endif
/*  Title:     Raw Audio Loader
 *  Version:   0.5
 *  Filename:  raw.h
 *  Authors:   Brett Mravec
 *  Purpose:   load raw data from file to buffer
 *  Date:      23 April 2009
 */

#ifndef __RAW_H__
#define __RAW_H__

char raw_init (void);
char raw_read_frame (void);

#endif /* __RAW_H__ */
/*  Title:     SD Card Interface
 *  Version:   1.0
 *  Filename:  sd.h
 *  Authors:   Brett Mravec, Greg McCoy
 *  Purpose:   Low-level SD card communication
 *  Date:      25 Mar 2009
 */

#ifndef __SD_H__
#define __SD_H__

// SPI related constants
#define SD_SPICTL (SPIMS|SPIEN|MSBF|TIMOD1|/*SMLS|*/CPHASE|CLKPL) // Not specifying a
wl defaults to 8
#define SD_INIT_BAUD 0x200
#define SD_BAUD 0x16

// SD Card Commands
#define CMD0_GO_IDLE_STATE            0x00
#define CMD1_SEND_OPCOND              0x01
#define CMD9_SEND_CSD                 0x09
#define CMD10_SEND_CID                0x0a
#define CMD12_STOP_TRANSMISSION       0x0b
#define CMD13_SEND_STATUS             0x0c
#define CMD16_SET_BLOCKLEN            0x10
#define CMD17_READ_SINGLE_BLOCK       0x11
#define CMD18_READ_MULTIPLE_BLOCK     0x12
#define CMD24_WRITE_BLOCK             0x18
#define CMD25_WRITE_MULTIPLE_BLOCK    0x19
#define CMD27_PROGRAM_CSD             0x1b
#define CMD28_SET_WRITE_PROT          0x1c
#define CMD29_CLR_WRITE_PROT          0x1d
#define CMD30_SEND_WRITE_PROT         0x1e
#define CMD32_ERASE_WR_BLK_START_ADDR 0x20
#define CMD33_ERASE_WR_BLK_END_ADDR   0x21
#define CMD38_ERASE                   0x26
#define CMD55_APP_CMD                 0x37
#define CMD56_GEN_CMD                 0x38
#define CMD58_READ_OCR                0x3a
#define CMD59_CRC_ON_OFF              0x3b
```

```
// Application-specific commands (always prefixed with CMD55_APP_CMD)
#define ACMD13_SD_STATUS              0x0d
#define ACMD22_SEND_NUM_WR_BLOCKS     0x16
#define ACMD23_SET_WR_BLK_ERASE_COUNT 0x17
#define ACMD41_SEND_OP_COND           0x29
#define ACMD42_SET_CLR_CARD_DETECT    0x2a
#define ACMD51_SEND_SCR               0x33

// R1 format responses (ORed together as a bit-field)
#define R1_NOERROR                    0x00
#define R1_IDLE                       0x01
#define R1_ERASE                      0x02
#define R1_ILLEGAL                    0x04
#define R1_CRC_ERR                    0x08
#define R1_ERASE_SEQ                  0x10
#define R1_ADDR_ERR                   0x20
#define R1_PARAM_ERR                  0x40

// R2 format responses - second byte only, first is identical to R1
#define R2_LOCKED                     0x01
#define R2_WP_FAILED                  0x02
#define R2_ERROR                      0x04
#define R2_CTRL_ERR                   0x08
#define R2_ECC_FAIL                   0x10
#define R2_WP_VIOL                    0x20
#define R2_ERASE_PARAM                0x40
#define R2_RANGE_ERR                  0x80

// CRC-related constants
#define SD_CRC7         0
#define SD_CRC16        1
#define CRC_OK          0
#define CRC_FAIL       -1

// Transfer-related return codes
#define TR_OK           0
#define TR_INVALID_ARG -1
#define TR_TIMEOUT     -2
#define TR_ERROR_TOKEN -3
#define TR_NOT_IDLE    -4
#define TR_FAILURE     -5

// Misc defines
#define BLOCK_BUFFER_LEN   515
#define INPUT_BUFFER_LEN   10
#define WAIT_R1_TIMEOUT    50
#define WAIT_WRITE_TIMEOUT 32768

// Function prototypes
int sd_init(void);
char sd_send_cmd (char command, int arg);
char sd_read_block (int addr, char *buff);

void sd_flush (void);

#endif /* __SD_H__ */
/*  Title:     SPI
 *  Version:   1.0
```

```
 *  Filename:  spi.h
 *  Authors:   Brett Mravec
 *  Purpose:   SPI communication routines
 *  Date:      6 Apr 2009
 */

#ifndef __SPI_H__
#define __SPI_H__

char spi_xfer (char b);

#endif /* __SPI_H__ */
/*  Title:     Global Utilities
 *  Version:   1.0
 *  Filename:  util.h
 *  Authors:   Greg McCoy, Isaac Jones, Brett Mravec
 *  Purpose:   Common utilities applicable to all software modules
 *  Date:      2 Apr 2009
 */

#ifndef __UTIL_H__
#define __UTIL_H__

#define TIME_PER_CYCLE 5

void delay_processor (int count);
void memcpy (char *dest, char *src, int len);
char ends_with (char *string, char *suffix);

#endif /* __UTIL_H__ */
/* Title:  Audio Interface
 * Version: 0.1
 * Filename:  audio.c
 * Authors:  Brett Mravec & Isaac Jones
 * Purpose:  Routines for DAI and AD1854
 * Date:  04/19/2009
 */

#include "audio.h"
#include "FlacTrac.h"
#include "util.h"

#include <Cdef21262.h>
#include <def21262.h>

#define NUM_BUFFERS 3
#define FRAME_SIZE 2304

#define AUDIO_CTL (MSTR | SPTRAN | SPEN_A | OPMODE | SLEN16 | SDEN_A)

typedef struct _AudioBuffer AudioBuffer;
struct _AudioBuffer {
    char full;
    short samples[FRAME_SIZE];
};

static int num_underrun;
static int tot_num;
```

```
static int total_file_samples;
static int total_elapsed_samples;

AudioBuffer buffer[NUM_BUFFERS];
static char buffer_in;
static char buffer_out;
static short temporary_buffer[FRAME_SIZE];

/* Function Name:  audio_init
 * Author(s):  Isaac Jones, Brett Mravec
 * Purpose:  Initializes Digital Audio Interface
 * Parameters:  total_file_ssamples; an integer that represents the total
 *              number of samples in the currently playing FLAC file.
 * Returns:  None.
 */

void
audio_init (int total_file_samples)
{
    int i = 0;
    interrupts(SIG_SP0, audio_sport_isr);

    num_underrun = 0;
    tot_num = 0;

    //AD1854 connections:
    //DAI Pin 1 SDATA
    //DAI Pin 2 MCLK
    //DAI Pin 3 BCLK
    //DAI Pin 4 L/nR CLK

    //Set up dai pins as outputs
    SRU(HIGH, PBEN01_I);
    SRU(HIGH, PBEN02_I);
    SRU(HIGH, PBEN03_I);
    SRU(HIGH, PBEN04_I);

    //Map the Serial port pins to the output pins on the DSP
    SRU(SPORT0_DA_O, DAI_PB01_I);
    SRU(PCG_CLKA_O, DAI_PB02_I);
    SRU(SPORT0_CLK_O, DAI_PB03_I);
    SRU(SPORT0_FS_O, DAI_PB04_I);

    //Set up the clock generator
    *pPCG_CTLA0 = ENCLKA;
    *pPCG_CTLA1 = 0x00;

    // Set Clock divisor and FS divisor for SP0
    *pDIV0 = 16 << 16 | 0x0044;

    *pSPCTL0 = 0;

    buffer_in = 0;
    buffer_out = -1;
    for (i = 0; i < NUM_BUFFERS; i++) {
        buffer[i].full = 0;
    }
```

```
}

/* Function Name: audio_get_progress
 * Author(s): Isaac Jones
 * Purpose: Return the progress as a percentage.  Uses
 *       total_file_samples and total_elapsed_samples.
 * Parameters: None.
 * Returns: An integer from 0 to 100.
 */

int
audio_get_progress (void)
{
    if( total_elapsed_samples == 0) {
        return 0;
    } else if (total_elapsed_samples == total_file_samples) {
        return 100;
    } else {
        return (int)100*((float)total_elapsed_samples/(float)total_file_samples);
    }
}

/* Function Name: audio_sport_isr
 * Author(s): Brett Mravec
 * Purpose: Give DMA the next block of memory to send out and update the buffer
 *          full status accordingly
 * Parameters: sig_int identifier for the interrupt that called the function
 * Returns: None.
 */

void
audio_sport_isr (int sig_int)
{
    int i = 0;

    *pSPCTL0 = 0;

    buffer[buffer_out].full = 0;

    buffer_out = (buffer_out + 1) % NUM_BUFFERS;

    tot_num++;
    if (buffer[buffer_out].full == 0) {
        buffer_out = -1;

        num_underrun++;

        return;
    }

    *pCSP0A = FRAME_SIZE;
    *pIMSP0A = 1;
    *pIISP0A = (unsigned int) buffer[buffer_out].samples;

    *pSPCTL0 = AUDIO_CTL;
}

/* Function Name: audio_buffer_set_full
```

```
 * Author(s): Brett Mravec
 * Purpose: Set active buffer as full and start DMA operation if not already
 *          one in progress
 * Parameters: None.
 * Returns: None.
 */

void
audio_buffer_set_full (void)
{
    int i = 0;
    short *temporary = NULL;

    buffer[buffer_in].full = 1;

    for(i = 0; i < FRAME_SIZE/2; i++) {
        temporary_buffer[2*i] = buffer[buffer_in].samples[i];
        temporary_buffer[2*i+1] = buffer[buffer_in].samples[i+FRAME_SIZE/2];

        temporary_buffer[2*i] = (temporary_buffer[2*i] / 4) + (65535 / 4);
        temporary_buffer[2*i+1] = (temporary_buffer[2*i+1] / 4) + (65535 / 4);
    }

    memcpy ((char *)buffer[buffer_in].samples, (char *)temporary_buffer, FRAME_SIZE);

    if (buffer_out == -1) {
        buffer_out = buffer_in;
        *pSPCTL0 = 0;

        *pCSP0A = FRAME_SIZE;
        *pIMSP0A = 1;
        *pIISP0A = (unsigned int) buffer[buffer_out].samples;

        *pSPCTL0 = AUDIO_CTL;
    }

    buffer_in++;
    buffer_in %= NUM_BUFFERS;
}

/* Function Name: audio_get_buffer
 * Author(s): Brett Mravec
 * Purpose: Return currently active buffer that the caller should fill with data
 * Parameters: None.
 * Returns: short*  buffer for the caller to fill
 */

short*
audio_get_buffer (void)
{
    int i = 0;

    if (buffer[buffer_in].full == 1) {
        return NULL;
    }

    return buffer[buffer_in].samples;
}
```

```
/* Function Name: audio_clear
 * Author(s): Brett Mravec
 * Purpose: Stop any DMA operations in progress and clear buffers
 * Parameters: None.
 * Returns: None.
 */

void
audio_clear (void)
{
    int i;

    buffer_in = 0;
    buffer_out = -1;

    *pSPCTL0 = 0;

    for (i = 0; i < NUM_BUFFERS; i++) {
        buffer[i].full = 0;
    }
}

/* Function Name: audio_get_samples
 * Author(s): Brett Mravec
 * Purpose: Fills given array with a representation of the samples currently in
 *          the output buffer being transmitted via DMA
 * Parameters: samples array to fill with data
 *             num_samples length of array to load
 * Returns: None.
 */

void
audio_get_samples (short *samples, int num_samples)
{
    int i, j = 0, num = FRAME_SIZE / num_samples;

    if (buffer_out == -1) {
        for (i = 0; i < num_samples; i++) {
            samples[i] = 0;
        }

        return;
    }

    int avg;
    for (i = 0; i < num_samples; i++) {
        avg = 0;
        do {
            avg = avg + 4 * (buffer[buffer_out].samples[j++] - 65535/4);
        } while (j % num);

        samples[i] = (avg / num) / 4096;
    }
}
/*  Title:     Bit Reader
 *  Version:   0.5
 *  Filename:  bit-reader.c
```

```
 *  Authors:   Brett Mravec
 *  Purpose:   Provides bit data stream
 *  Date:      10 April 2009
 */

#include "bit-reader.h"
#include "fat.h"

void
bit_reader_init (BitReader *br)
{
    br->b = fat_getc ();
    br->bit_pos = 0x80;
}

int
bit_reader_get (BitReader *br, char bits)
{
    int val = 0;

    while (bits > 0) {
        if (br->b & br->bit_pos)
            val |= (1 << bits-1);

        bits--;
        br->bit_pos >>= 1;

        if (!br->bit_pos) {
            br->b = fat_getc ();
            br->bit_pos = 0x80;
        }
    }

    return val;
}

int
bit_reader_get_s (BitReader *br, char bits)
{
    int val = 0;

    if (br->b & br->bit_pos)
        val = 0xffffffff;

    while (bits > 0) {
        if (!(br->b & br->bit_pos))
            val &= ~(1 << bits-1);
        else
            val |= 1 << bits-1;

        bits--;
        br->bit_pos >>= 1;

        if (!br->bit_pos) {
            br->b = fat_getc ();
            br->bit_pos = 0x80;
        }
    }
```

```
    return val;
}


int
bit_reader_get_unary (BitReader *br)
{
    int value = 0;

    while (!(br->b & br->bit_pos)) {
        value++;

        br->bit_pos >>= 1;

        if (!br->bit_pos) {
            br->b = fat_getc ();
            br->bit_pos = 0x80;
        }
    }

    br->bit_pos >>= 1;
    if (!br->bit_pos) {
        br->b = fat_getc ();
        br->bit_pos = 0x80;
    }

    return value;
}

int
bit_reader_align (BitReader *br)
{
    int cnt = 0;

    if (br->bit_pos != 0x80) {
        while (br->bit_pos) {
            br->bit_pos >>= 1;
            cnt++;
        }

        br->bit_pos = 0x80;
        br->b = fat_getc ();
    }

    return cnt;
}

long
bit_reader_skip (BitReader *br, long bits)
{
    //TODO: Possible optimization here
//    bits -= bit_reader_align (br);
    while (br->bit_pos) {
        br->bit_pos >>= 1;
        bits--;
    }
```

```
    bits -= 8 * fat_skip (bits / 8);

    if (bits >= 8) {
        return bits;
    }

    br->bit_pos = 0x80;
    br->b = fat_getc ();

    bit_reader_get (br, bits % 8);
    bits -= bits % 8;

    return bits;
}

// Modified version of FLAC__bitbuffer_read_utf8_uint64 from bitbuffer.c in flac-1.1.0
long
bit_reader_get_utf8 (BitReader *br)
{
    long v = 0;
    int x;
    unsigned i;

    x = bit_reader_get (br, 8);

    if(!(x & 0x80)) { /* 0xxxxxxx */
        v = x;
        i = 0;
    } else if (x & 0xC0 && !(x & 0x20)) { /* 110xxxxx */
        v = x & 0x1F;
        i = 1;
    } else if (x & 0xE0 && !(x & 0x10)) { /* 1110xxxx */
        v = x & 0x0F;
        i = 2;
    } else if (x & 0xF0 && !(x & 0x08)) { /* 11110xxx */
        v = x & 0x07;
        i = 3;
    } else if (x & 0xF8 && !(x & 0x04)) { /* 111110xx */
        v = x & 0x03;
        i = 4;
    } else if (x & 0xFC && !(x & 0x02)) { /* 1111110x */
        v = x & 0x01;
        i = 5;
    } else if (x & 0xFE && !(x & 0x01)) { /* 11111110 */
        v = 0;
        i = 6;
    } else {
        return 0xffffffff;
    }

    for( ; i; i--) {
        x = bit_reader_get (br, 8);

        if(!(x & 0x80) || (x & 0x40)) { /* 10xxxxxx */
            return (long) 0xffffffff;
        }

        v <<= 6;
```

```
        v |= (x & 0x3F);
    }
    return v;
}

int
bit_reader_get_rice (BitReader *br, int M)
{
    int Q;//, R;

//    Q = bit_reader_get_unary (&br) << M;
//    Q |= bit_reader_get (&br, M);

    Q = bit_reader_get_unary (br) << M |
        bit_reader_get (br, M);

//    Q <<= M;
//    Q |= R;

    return (Q >> 1) ^ -(Q & 1);
}
/*  Title:     FAT Filesystem
 *  Version:   1.0
 *  Filename:  fat.c
 *  Authors:   Brett Mravec
 *  Purpose:   FAT filesystem routines for SD card
 *  Date:      28 Mar 2009
 */

#include "fat.h"
#include "sd.h"
#include "util.h"

static int fs_start;  // filesystem starting sector
static int fat_start; // file allocation table starting sector
static int dr_start;  // sector of data section

char file_buff[512]; // read buffer
char temp_buff[512];

static short bps; // bytes per sector
static char spc; // sector per cluster
static int spfat; // sectors per fat

static char open_byte;   // byte offset in sector of open file
static char open_sector; // sector offset in cluster of open file
static int open_cluster; // cluster of the open file
static char eof_flag;

static int open_size;    // number of byte total in open file
static int open_read;    // number of bytes read from open file

char fat_cache[512];
static int fat_page;

FileEntry files[FAT_NUM_ENTRIES];
static int file_offset;
```

```
static FileEntry current_dir;
static FileEntry current_file;

void
fat_read_block (int addr, char *buff)
{
    int status;
    do {
        status = sd_read_block (addr, buff);

        if (status == -1) {
            sd_flush ();
        } else if (status == 0x08) {
            return;
        }
    } while (status == -1);
}

int
fat_get_next (int cluster)
{
    int page = (4 * cluster) / bps;
    int off = (4 * cluster) % bps;

    if (cluster == 0 || cluster == 1) {
        return 0x0ffffffe;
    }

    if (page != fat_page) {
        fat_page = page;
        fat_read_block ((fat_start + page) * bps, fat_cache);

    }

    return fat_cache[off]         |
           fat_cache[off+1] << 8  |
           fat_cache[off+2] << 16 |
           fat_cache[off+3] << 24;
}

void
fat_load_dir (int cluster)
{
    int i, j, k = 0;
    FileEntry fe;
    char finished = 0;
    int status;

    while (!finished) {
        for (i = 0; i < spc && !finished; i++) {
            fat_read_block ((dr_start + (cluster - 2) * spc + i) * bps, temp_buff);

            for (j = 0; j < 16 && !finished; j++) {
                if (temp_buff[j*32] == 0) {
                    finished = 1;
                    files[k - file_offset].name[0] = '\0';
                    break;
                } else if (temp_buff[j*32] == 0xe5) {
```

```
                    continue;
                } else if (temp_buff[j*32+0x0b] == 0x0f) {
                    char sn = temp_buff[j*32];

                    if (sn & 0x80) {
                        continue;
                    }

                    fe.name[((0x3f & sn) - 1) * 13 + 0] = temp_buff[j*32+0x01];
                    fe.name[((0x3f & sn) - 1) * 13 + 1] = temp_buff[j*32+0x03];
                    fe.name[((0x3f & sn) - 1) * 13 + 2] = temp_buff[j*32+0x05];
                    fe.name[((0x3f & sn) - 1) * 13 + 3] = temp_buff[j*32+0x07];
                    fe.name[((0x3f & sn) - 1) * 13 + 4] = temp_buff[j*32+0x09];

                    fe.name[((0x3f & sn) - 1) * 13 + 5] = temp_buff[j*32+0x0e];
                    fe.name[((0x3f & sn) - 1) * 13 + 6] = temp_buff[j*32+0x10];
                    fe.name[((0x3f & sn) - 1) * 13 + 7] = temp_buff[j*32+0x12];
                    fe.name[((0x3f & sn) - 1) * 13 + 8] = temp_buff[j*32+0x14];
                    fe.name[((0x3f & sn) - 1) * 13 + 9] = temp_buff[j*32+0x16];
                    fe.name[((0x3f & sn) - 1) * 13 + 10] = temp_buff[j*32+0x18];

                    fe.name[((0x3f & sn) - 1) * 13 + 11] = temp_buff[j*32+0x1c];
                    fe.name[((0x3f & sn) - 1) * 13 + 12] = temp_buff[j*32+0x1e];

                    if (sn & 0x40) {
                        fe.name[(0x3f & sn) * 13 + 13] = '\0';
                    }
                } else if (temp_buff[j*32+0x0b] & 0x08) {
                    continue;
                } else if (temp_buff[j*32+0x0b] & 0x02) {
                    continue;
                } else {
                    if (temp_buff[j*32] == 0x2e) {
                        fe.name[0] = '.';
                        fe.name[1] = '.';
                        fe.name[2] = '\0';
                    }

                    fe.cluster = temp_buff[j*32+0x1a];
                    fe.cluster |= temp_buff[j*32+0x1b] << 8;
                    fe.cluster |= temp_buff[j*32+0x14] << 16;
                    fe.cluster |= temp_buff[j*32+0x15] << 24;

                    fe.size = temp_buff[j*32+0x1c];
                    fe.size |= temp_buff[j*32+0x1d] << 8;
                    fe.size |= temp_buff[j*32+0x1e] << 16;
                    fe.size |= temp_buff[j*32+0x1f] << 24;

                    if (k >= file_offset) {
                        memcpy ((char*) &files[k - file_offset], (char*) &fe, sizeof
(FileEntry));
                    }

                    k++;
                    if (k >= file_offset + FAT_NUM_ENTRIES) {
                        finished = 1;
                    }
                }
```

```
            }
        }

        int next = fat_get_next (cluster);

        if (next >= 0x0ffffff8) {
            finished = 1;
        } else {
            cluster = next;
        }
    }
}

void
fat_init (void)
{
    char status;
    fat_page = -1;
    open_cluster = -1;
    file_offset = 0;

    fat_read_block (0, temp_buff);

    fs_start = temp_buff[454];
    fs_start |= temp_buff[455] << 8;
    fs_start |= temp_buff[456] << 16;
    fs_start |= temp_buff[457] << 24;

    fat_read_block (512 * fs_start, temp_buff);

    bps = temp_buff[0x0b];
    bps |= temp_buff[0x0c] << 8;

    spc = temp_buff[0x0d];

    spfat = temp_buff[0x24];
    spfat |= temp_buff[0x25] << 8;
    spfat |= temp_buff[0x26] << 16;
    spfat |= temp_buff[0x27] << 24;

    // first fat starts at end of reserved sectors of partition
    fat_start = temp_buff[0x0e];
    fat_start |= temp_buff[0x0f] << 8;
    fat_start += fs_start;

    // first directory starts after last fat
    dr_start = fat_start + temp_buff[0x10] * spfat;

    current_dir.name[0] = '/';
    current_dir.name[1] = '\0';
    current_dir.size = 0;
    current_dir.cluster = temp_buff[0x2c] |
        temp_buff[0x2d] << 8 |
        temp_buff[0x2e] << 16 |
        temp_buff[0x2f] << 24;

    fat_load_dir (
        temp_buff[0x2c] |
```

```
        temp_buff[0x2d] << 8 |
        temp_buff[0x2e] << 16 |
        temp_buff[0x2f] << 24);
}

void
fat_open (FileEntry *fptr)
{
    memcpy ((char*) &current_file, (char*) fptr, sizeof (FileEntry));

    open_cluster = fptr->cluster;
    open_sector = 0;
    open_byte = 0;
    open_size = fptr->size;
    open_read = 0;
    eof_flag = 0;

    fat_read_block ((dr_start + (open_cluster - 2) * spc) * bps, file_buff);
}

void
fat_close (void)
{
    int i;
    for (i = 0; i < 512; i++) {
        file_buff[i] = 0;
    }

    open_cluster = -1;
    open_sector = 0;
}

int
fat_skip (int len)
{
    int i;

    if (eof_flag) {
        return -1;
    }

    for (i = 0; i < len; i++) {
        if (open_read == open_size) {
            eof_flag = 1;
            break;
        }

        open_byte++;
        open_read++;

        if (open_byte >= bps) {
            open_sector++;
            open_byte = 0;

            if (open_sector >= spc) {
                int nc = fat_get_next (open_cluster);

                if (nc >= 0x0ffffff8) {
```

```
                    eof_flag = 1;
                    break;
                } else {
                    open_cluster = nc;
                    open_sector = 0;
                }
            }

//            fat_read_block ((dr_start + (open_cluster - 2) * spc + open_sector) *
bps, file_buff);
        }
    }

    fat_read_block ((dr_start + (open_cluster - 2) * spc + open_sector) * bps,
file_buff);

    return i;
}

int
fat_getc (void)
{
    if (eof_flag) {
        return -1;
    }

    if (open_read >= open_size) {
        eof_flag = 1;
        return -1;
    }

    int value = file_buff[open_byte++];
    open_read++;

    if (open_byte >= bps) {
        open_sector++;
        open_byte = 0;

        if (open_sector >= spc) {
            int nc = fat_get_next (open_cluster);

            if (nc >= 0x0ffffff8) {
                eof_flag = 1;
            } else {
                open_cluster = nc;
                open_sector = 0;
            }
        }

        if (!eof_flag)
            fat_read_block ((dr_start + (open_cluster - 2) * spc + open_sector) * bps,
file_buff);
    }

    return value;
}

int
```

```
fat_read (char *buff, int len)
{
    int i;

    if (eof_flag) {
        return -1;
    }

    for (i = 0; i < len; i++) {
        if (eof_flag) {
            break;
        }
/*
        buff[i] = file_buff[open_byte++];
        open_read++;

        if (open_byte >= bps) {
            open_sector++;
            open_byte = 0;

            if (open_sector >= spc) {
                int nc = fat_get_next (open_cluster);

                if (nc >= 0x0ffffff8) {
                    eof_flag = 1;
                    break;
                } else {
                    open_cluster = nc;
                    open_sector = 0;
                }
            }

            fat_read_block ((dr_start + (open_cluster - 2) * spc + open_sector) * bps,
file_buff);
        } */

        buff[i] = fat_getc ();
    }

    return i;
}

char
fat_eof (void)
{
    return eof_flag;
}

FileEntry*
fat_ls (void)
{
    return files;
}

void
fat_ls_down (void)
{
    file_offset++;
```

```
    fat_load_dir (current_dir.cluster);

    int i;
    for (i = 0; i < FAT_NUM_ENTRIES; i++)
        if (files[i].name[0] == '\0')
            break;

    if (i < FAT_NUM_ENTRIES) {
        file_offset--;
        fat_load_dir (current_dir.cluster);
    }
}

void
fat_ls_up (void)
{
    if (file_offset != 0) {
        file_offset--;
        fat_load_dir (current_dir.cluster);
    }
}

void
fat_cd (FileEntry *fptr)
{
    if (fptr->type == TYPE_FILE)
        return;

    memcpy ((char*) &current_dir, (char*) fptr, sizeof (FileEntry));
    file_offset = 0;

    // Add checks for valid cluster
    fat_load_dir (current_dir.cluster);
}

FileEntry*
fat_pwd (void)
{
    return &current_dir;
}
/*  Title:     Flac decoder
 *  Version:   0.5
 *  Filename:  flac.c
 *  Authors:   Brett Mravec
 *  Purpose:   State based flac decoder
 *  Date:      10 April 2009
 */

#include "flac.h"
#include "bit-reader.h"
#include "fat.h"
#include "audio.h"

#include <stdio.h>

typedef struct _SubframeHeader SubframeHeader;
struct _SubframeHeader {
```

```
    char type;
    char wasted_bps;
};

typedef struct _FrameHeader FrameHeader;
struct _FrameHeader {
    char sync_byte;
    char block_strategy;
    short block_size;
    int sample_rate;
    char channels;
    char sample_size;
    long fs_number;
    char crc8;
};

char flac_read_frame_header (void);
void flac_read_subframe_header (void);

void flac_read_subframe_constant (short *data, int bps);
void flac_read_subframe_verbatim (short *data, int bps);
void flac_read_subframe_fixed (short *data, int bps);
void flac_read_subframe_lpc (short *data, int bps);
void flac_read_residual (short *data, char predictor_order);

static BitReader br;

// VORBIS COMMENT
static char tag_key[3][7] = { "ARTIST", "ALBUM", "TITLE" };
static char tag_val[3][22];

#define ARTIST_TAG 0
#define ALBUM_TAG 1
#define TITLE_TAG 2
// END VORBIS COMMENT

// STREAMINFO
static short block_min;
static short block_max;

static int frame_min;
static int frame_max;

static int sample_rate;
static char channels;
static char bpsample;

static long total_samples;
static long read_samples;
// END STREAMINFO

// DECODE HEADERS
static SubframeHeader sfh;
static FrameHeader fh;
// END DECODE HEADERS

static char num_channels[16] = {
    1, 2, 3, 4, 5, 6, 7, 8, 2, 2, 2, -1, -1, -1, -1, -1
```

```
};

static int block_sizes[16] = {
    -1, 192, 574, 1152, 2304, 4608, 6, 7, 256, 512,
    1024, 2048, 4096, 8192, 16384, 32768
};

static int sample_rates[16] = {
    0, 88200, 176400, 192000, 8000, 16000, 22050,
    24000, 32000, 44100, 48000, 96000, 12, 13, 14, 15
};

static char bitspersample[8] = {
   0, 8, 12, -1, 16, 20, 24, -1
};

int
flac_read_vorbis_int (void)
{
    return bit_reader_get (&br, 8)       |
           bit_reader_get (&br, 8) <<  8 |
           bit_reader_get (&br, 8) << 16 |
           bit_reader_get (&br, 8) << 24;
}

char
flac_strcmp (char *str1, char *str2)
{
    int i;
    for (;;i++) {
        if (str1[i] == '\0' && str2[i] == '\0')
            break;
        if (str1[i] == '\0') return 1;
        if (str2[i] == '\0') return -1;
        if (str1[i] < str2[i]) return 1;
        if (str2[i] < str1[i]) return 1;
    }

    return 0;
}

int
flac_read_metadata (void)
{
    char last;
    char type;
    int length, tlen, num_tags, i, j;
    char *dest;
    char key[8];

    tag_val[0][0] = '\0';
    tag_val[1][0] = '\0';
    tag_val[2][0] = '\0';

    do {
        last = bit_reader_get (&br, 1);
        type = bit_reader_get (&br, 7);
        length = bit_reader_get (&br, 24);
```

```
if (type >= 7) {
    return -1;
}

switch (type) {
    case 0: // STREAMINFO
        block_min = bit_reader_get (&br, 16);
        block_max = bit_reader_get (&br, 16);

        frame_min = bit_reader_get (&br, 24);
        frame_max = bit_reader_get (&br, 24);

        sample_rate = bit_reader_get (&br, 20);
        channels = bit_reader_get (&br, 3) + 1;
        bpsample = bit_reader_get (&br, 5) + 1;

        total_samples = bit_reader_get (&br, 4);
        total_samples <<= 32;
        total_samples |= bit_reader_get (&br, 32);

        bit_reader_skip (&br, 128);
        break;
    case 4: // VORBIS_COMMENT
        tlen = flac_read_vorbis_int ();
        bit_reader_skip (&br, 8*tlen);

        num_tags = flac_read_vorbis_int ();
        for (;num_tags > 0; num_tags--) {
            dest = 0;

            tlen = flac_read_vorbis_int ();
            for (j = 0; j < tlen; j++) {
                key[j] = bit_reader_get (&br, 8);
                if (key[j] == '=' || j == 7) {
                    key[j] = '\0';
                    break;
                }
            }

            for (i = 0; i < 3; i++) {
                if (!flac_strcmp (tag_key[i], key)) {
                    dest = tag_val[i];
                    break;
                }
            }

            if (!dest) {
                bit_reader_skip (&br, 8*(tlen-j-1));
            } else {
                for (i = 0; i < tlen - j - 1; i++) {
                    dest[i] = bit_reader_get (&br, 8);
                    if (i == 20) {
                        dest[21] = '\0';
                        bit_reader_skip (&br, 8*(tlen-j-i-1));
                        break;
                    }
                }
```

```
                        if (i <= tlen - j - 1)
                            dest[i] = '\0';
                    }
                }
                break;
            default:
                bit_reader_skip (&br, 8*length);
        }
    } while (!last);

    return 0;
}

char
flac_init (void)
{
    char i;

    bit_reader_init (&br);

    if (bit_reader_get (&br, 8) != 'f') return -1;
    if (bit_reader_get (&br, 8) != 'L') return -1;
    if (bit_reader_get (&br, 8) != 'a') return -1;
    if (bit_reader_get (&br, 8) != 'C') return -1;

    if (flac_read_metadata ()) {
        return -2;
    }

    if (block_min != 1152 || block_max != 1152) {
        return -1;
    }

    audio_init (total_samples);

    return 0;
}

char
flac_read_frame (void)
{
    short *frame;
    char channel, channels;
    int i, bps;

    frame = audio_get_buffer ();

    if (!frame) {
        return 0;
    }

    if (flac_read_frame_header ()) {
        return -1;
    }

    channels = num_channels[fh.channels];
    for (channel = 0; channel < channels; channel++) {
```

```
        flac_read_subframe_header ();

        bps = fh.sample_size;
        if ((fh.channels == 0x08 || fh.channels == 0x0a) && channel == 1)
            bps++;
        else if (fh.channels == 0x09 && channel == 0)
            bps++;

        if (sfh.type & 0x20) {
            if (channel < 2) {
                flac_read_subframe_lpc (&frame[block_min*channel], bps);
            } else {
                flac_read_subframe_lpc (0, bps);
            }
        } else if (sfh.type & 0x10) {
            // RESERVED
        } else if (sfh.type & 0x08) {
            if (channel < 2) {
                flac_read_subframe_fixed (&frame[block_min*channel], bps);
            } else {
                flac_read_subframe_fixed (0, bps);
            }
        } else if (sfh.type & 0x04) {
            // RESERVED
        } else if (sfh.type & 0x02) {
            // RESERVED
        } else if (sfh.type & 0x01) {
            if (channel < 2) {
                flac_read_subframe_verbatim (&frame[block_min*channel], bps);
            } else {
                flac_read_subframe_verbatim (0, bps);
            }
        } else {
            if (channel < 2) {
                flac_read_subframe_constant (&frame[block_min*channel], bps);
            } else {
                flac_read_subframe_constant (0, bps);
            }
        }
    }

    if (fh.channels == 0x08) {
        for (i = 0; i < fh.block_size; i++)
            frame[block_min+i] = frame[i] - frame[block_min+i];
    } else if (fh.channels == 0x09) {
        for (i = 0; i < fh.block_size; i++)
            frame[i] += frame[block_min+i];
    } else if (fh.channels == 0x0a) {
        for (i = 0; i < fh.block_size; i++) {
            int mid = frame[i];
            int side = frame[block_min+i];

            mid <<= 1;
            if ((side & 1) != 0) {
                mid++;
            }

            int left = mid + side;
```

```
            int right = mid - side;

            frame[i] = left >> 1;
            frame[block_min+i] = right >> 1;
        }
    }

    audio_buffer_set_full ();

    return fat_eof ();
}

char*
flac_get_title (void)
{
    return tag_val[TITLE_TAG];
}

char*
flac_get_album (void)
{
    return tag_val[ALBUM_TAG];
}

char*
flac_get_artist (void)
{
    return tag_val[ARTIST_TAG];
}

char
flac_decode_progress (void)
{
    return (read_samples * 100) / total_samples;
}

char
flac_read_frame_header (void)
{
/*    do {
        if (fat_eof ()) return;
        bit_reader_align (&br);
        while ((fh.sync_byte = bit_reader_get (&br, 8)) != 0xff)
            if (fat_eof ()) return;
    } while ((fh.sync_byte = bit_reader_get (&br, 6)) != 0x3e);
*/
    char syncing = 1, sync_count = 0;
    while (syncing) {
        bit_reader_align (&br);

        while ((fh.sync_byte = bit_reader_get (&br, 8)) != 0xff) {
            if (sync_count++ > 1000)
                return -1;

            if (fat_eof ()) return -1;
        }

        while (2) {
```

```
            if (sync_count++ > 1000)
                return -1;

            if (fat_eof ()) return -1;

            fh.sync_byte = bit_reader_get (&br, 6);
            if (fh.sync_byte == 0x3e) {
                syncing = 0;
                break;
            } else {
                fh.sync_byte <<= 2;
                fh.sync_byte |= bit_reader_get (&br, 2);

                if (fh.sync_byte != 0xff)
                    break;
            }
        }
    }

    bit_reader_get (&br, 1);
    fh.block_strategy = bit_reader_get (&br, 1);

    fh.block_size = block_sizes[bit_reader_get (&br, 4)];
    fh.sample_rate = sample_rates[bit_reader_get (&br, 4)];

    fh.channels = bit_reader_get (&br, 4);

    fh.sample_size = bitspersample[bit_reader_get (&br, 3)];

    bit_reader_get (&br, 1);

    fh.fs_number = bit_reader_get_utf8 (&br);

    if (fh.block_size == 6) {
        fh.block_size = bit_reader_get (&br, 8);
    } else if (fh.block_size == 7) {
        fh.block_size = bit_reader_get (&br, 16);
    }

    if (fh.sample_rate == 12) {
        fh.sample_rate = 1000 * bit_reader_get(&br, 8);
    } else if (fh.sample_rate == 13) {
        fh.sample_rate = bit_reader_get (&br, 16);
    } else if (fh.sample_rate == 14) {
        fh.sample_rate = 10 * bit_reader_get (&br, 16);
    }

    fh.crc8 = bit_reader_get (&br, 8); // crc-8

    return 0;
}

void
flac_read_subframe_header (void)
{
    bit_reader_get (&br,1);

    sfh.type = bit_reader_get (&br, 6);
```

```
    sfh.wasted_bps = 0;

    if (bit_reader_get (&br, 1)) {
        sfh.wasted_bps = bit_reader_get_unary (&br);
        sfh.wasted_bps++;
    }
}

void
flac_read_subframe_constant (short *data, int bps)
{
    int i, sample;

    sample = bit_reader_get (&br, bps);

    if (data) {
        for (i = 0; i < fh.block_size; i++) {
            data[i] = sample;
        }
    }
}

void
flac_read_subframe_verbatim (short *data, int bps)
{
    int i;
    for (i = 0; i < fh.block_size; i++) {
        if (data) {
            data[i] = bit_reader_get (&br, bps);
        } else {
            bit_reader_get (&br, fh.sample_size);
        }
    }
}

void
flac_read_subframe_fixed (short *data, int bps)
{
    char order = 0x07 & sfh.type;
    int i, a, b, c, d;

    for (i = 0; i < order; i++) {
        if (data) {
            data[i] = bit_reader_get_s (&br, bps);
        } else {
            bit_reader_get_s (&br, bps);
        }
    }

    flac_read_residual (data, order);

    if (data) {
        a = data[order-1];
        b = a - data[order-2];
        c = b - data[order-2] + data[order-3];
        d = c - data[order-2] + 2*data[order-3] - data[order-4];

        switch(order) {
```

```
            case 1:
                for (i = order; i < fh.block_size; i++)
                    data[i] += a += data[i];
                break;
            case 2:
                for (i = order; i < fh.block_size; i++)
                    data[i] += a += b += data[i];
                break;
            case 3:
                for (i = order; i < fh.block_size; i++)
                    data[i] += a += b += c += data[i];
                break;
            case 4:
                for (i = order; i < fh.block_size; i++)
                    data[i] += a += b += c += d += data[i];
                break;
        }
    }
}

void
flac_read_subframe_lpc (short *data, int bps)
{
    char coeff_precision;
    char coeff_shift;
    int coeffs[32];
    int i;

    int order = (0x1f & sfh.type) + 1;

    for (i = 0; i < order; i++) {
        if (data) {
            data[i] = bit_reader_get_s (&br, bps);
        } else {
            bit_reader_get_s (&br, bps);
        }
    }

    coeff_precision = bit_reader_get (&br, 4) + 1;
    coeff_shift = bit_reader_get_s (&br, 5);

    for (i = 0; i < order; i++) {
        coeffs[i] = bit_reader_get_s (&br, coeff_precision);
    }

    flac_read_residual (data, order);

    if (data) {
        int j;
        for (i = order; i < fh.block_size; i++) {
            long sum = 0;
            for (j = 0; j < order; j++)
                sum += (long) coeffs[j] * (long) data[i - j - 1];
            data[i] += (short) (sum >> coeff_shift);
        }
    }

}
```

```
void
flac_read_residual (short *data, char predictor_order)
{
    char type = bit_reader_get (&br, 2);
    char partition_order = bit_reader_get (&br, 4);
    char r_param;
    int pos = predictor_order;
    int i, k, n;

    for (k = 0; k < 1 << partition_order; k++) {
        int fixed_sample_size = 0;
        if (type == 0) {
            r_param = 0x0f & bit_reader_get (&br, 4);
            if (r_param == 0xf)
                fixed_sample_size = 0x1f & bit_reader_get (&br, 5);
        } else if (type == 1) {
            r_param = 0x1f & bit_reader_get (&br, 5);
            if (r_param == 0x1f)
                fixed_sample_size = 0x1f & bit_reader_get (&br, 5);
        }

        n = 0;
        if (k == 0) {
            n = (fh.block_size >> partition_order) - predictor_order;
        } else if (partition_order == 0) {
            n = fh.block_size - predictor_order;
        } else {
            n = fh.block_size >> partition_order;
        }

        if (fixed_sample_size == 0) {
            for (i = 0; i < n; i++) {
                if (data) {
                    data[pos++] = bit_reader_get_rice (&br, r_param);
                } else {
                    bit_reader_get_rice (&br, r_param);
                }
            }
        } else {
            for (i = 0; i < n; i++) {
                if (data) {
                    data[pos++] = bit_reader_get (&br, fixed_sample_size);
                } else {
                    bit_reader_get (&br, fixed_sample_size);
                }
            }
        }
    }
}
/* Title: LCD Control Functions
 * Version: 0.10
 * Filename: lcd.c
 * Author(s): Isaac Jones & Greg McCoy & Brett Mravec
 * Purpose/Function of Program:
 *      Contains functions and global variables that control
 *      the display of user data on the LCD.  This includes
 *      the two main User Interfaces, and any dynamic updating
```

```
 *      that needs to be done.  With the functions contained
 *      herein, no functions outside of this file should need
 *      to access the LCD screen directly.
 * How Program is Run on Target System:  Not controlled by user.
 *      Code is compiled and loaded to the DSP automatically.
 * Date Started: 03/31/2009
 * Update History:
 *  03/31/2009: Started file.  Added basic functions to try to
 *      interface with the LCD.
 *  04/01/2009: Changed the way some of the functions worked.
 *      Most notably, lcd_send_cmd was changed to accept
 *      arbitrary input.
 *  04/02/2009: Added an LCD heartbeat function,
 *      lcd_toggle_displays.
 *  04/03/2009: Added various other LCD functions to attept to
 *      gain more functionality.
 *  04/05/2009: Added more testing functions.  The new LCD works
 *      in a semi-stable fashion.
 *  04/06/2009: Coded lcd_send_cmd in such a way as to send well-
 *      formed commands despite the SPI oddities.
 *  04/07/2009: Added lcd_print_directories.  1/2 of the "UI" is
 *      now complete.  Removed obsolete functions.
 *  04/08/2009: Added lcd_print_playback. "UI" now complete with
 *      very minor revisions possibly needed in the future.
 *  04/09/2009: Minor modifications to code to prepare for integration
 *      with the rest of the project code.  Added input sanity checking
 *      to lcd_print_directories.
 *  04/15/2009: Changed some LCD function to interface better with the
 *       main loop.
 */

#include "lcd.h"
#include "FlacTrac.h"
#include "spi.h"
#include "util.h"
#include "flac.h"
#include "audio.h"

#include <Cdef21262.h>
#include <def21262.h>
#include <sru21262.h>
#include <sysreg.h>

static int lcd_buffer[LCD_HEIGHT/CHAR_HEIGHT][LCD_WIDTH/4];
static char lcd_cursor_x;
static char lcd_cursor_y;
static short display_samples[128] ={ 0, 4114, 8213, 12280, 16297, 20251, 24125, 27903,
31571, 35115, 38520, 41773, 44861, 47772, 50495, 53018, 55333, 57428, 59297, 60932,
62327, 63476, 64374, 65018, 65405, 65535, 65405, 65018, 64374, 63476, 62327, 60932,
59297, 57428, 55333, 53018, 50495, 47772, 44861, 41773, 38520, 35115, 31571, 27903,
24125, 20251, 16297, 12280, 8213, 4114, 0, -4114, -8213, -12280, -16297, -20251, -
24125, -27903, -31571, -35115, -38520, -41773, -44861, -47772, -50495, -53018, -55333,
-57428, -59297, -60932, -62327, -63476, -64374, -65018, -65405, -65535, -65405, -
65018, -64374, -63476, -62327, -60932, -59297, -57428, -55333, -53018, -50495, -47772,
-44861, -41773, -38520, -35115, -31571, -27903, -24125, -20251, -16297, -12280, -8213,
-4114, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0};
```

```
/* Name: lcd_send_cmd
 * Author: Isaac Jones & Greg McCoy
 * Called By: lcd_init(), lcd_update()
 * Function: Sends a single command to the LCD module.
 *      A single command can Write Data, Change the X value
 *      (the page undergoing operations), Change the Y value
 *      (the byte undergoing operations), Change the Z value
 *      (the starting line of the LCD  (not used)), or
 *      change the display state of a column.  Any of these
 *      operations can be performed on one or both columns.
 *      Input checking is performed to ensure that the sent byte
 *      is valid data for the type of command issued.
 * Usage: lcd_send_cmd(
 *          type (integer, one of WRITE_DATA, SET_X, SET_Y, SET_Z, SET_POWER)
 *          columns (integer, one of CS1, CS2, BOTH_COLUMNS)
 *          byte (data to be send, any 8-bit sequence)
 * Changes to state:  Changes the state of the SPI transmit buffer.
 *          It will return an empty transmit buffer regardless of the
 *          data when called.  Also changes the state of the flag pins
 *          to transmit the proper commands to the LCD.  The state of
 *          these pins on return is unpredictable.
*/

void
lcd_send_cmd(int type, int columns, char byte)
{
    char send_byte = 0;

    // Set up the byte to ensure that valid input is well-formed
    if (type == WRITE_DATA) {
        send_byte = byte;
    } else if (type == SET_Y) {
        send_byte = ((byte & 0x3F)|0x40);
    } else if (type == SET_X) {
        send_byte = ((byte & 0x07)|0xB8);
    } else if (type == SET_Z) {
        send_byte = ((byte & 0x3F)|0xC0);
    } else if (type == SET_POWER) {
        send_byte = ((byte & 0x01)|0x3E);
    } else {
        return;
    }

    delay_processor(E_LOW_BEFORE_ADDRESS);

    // Control CS1 & CS2 NOTE: Will be active low on actual board
    // NOTE: As of 4/5 the CS1 and CS2 active low switch was
    // accomplished in lcd_ctrl.  DO NOT CHANGE HERE.
    if (columns == CS1) {
        lcd_ctrl(CS1, 1);
        lcd_ctrl(CS2, 0);
    } else if (columns == CS2) {
        lcd_ctrl(CS1, 0);
        lcd_ctrl(CS2, 1);
    } else if (columns == BOTH_COLUMNS) {
        lcd_ctrl(CS1, 1);
        lcd_ctrl(CS2, 1);
    } else {
```

```
        lcd_ctrl(CS1, 0);
        lcd_ctrl(CS2, 0);
    }

    if (type == WRITE_DATA) {
        lcd_ctrl(RW, 0);
        lcd_ctrl(DI, 1);
    } else {
        lcd_ctrl(RW, 0);
        lcd_ctrl(DI, 0);
    }

    delay_processor(ADDRESS_SETUP_TIME);
    // Asert E to latch "address" data (control signals)
    lcd_ctrl(E, 1);

    delay_processor(E_HIGH_BEFORE_DATA);

    // Transfer the data on the SPI.
    spi_xfer(send_byte);

    delay_processor(DATA_SETUP_TIME);

    // De-assert E to latch data
    lcd_ctrl(E, 0);

    delay_processor(DATA_HOLD_TIME);
}

/* Name: lcd_ctrl
 * Author: Isaac Jones & Greg McCoy
 * Called By: lcd_send_cmd, lcd_init
 * Function: Changes the state of the given input
 *      pin to the given state.
 * Usage:  lcd_ctrl(
 *      cmd (integer, represents the pin to change.
 *          can be E, CS1, CS2, RW, DI, or SR)
 *      value (integer, represents the state to
 *          change to.  Can be 0 or n.)
 * Changes to state:  Changes the state of the
 *      specified pin.  No other state change.
*/

void
lcd_ctrl(char cmd, char value)
{
    /**************************************
    *
    *  4/5:  Changed to the "production" LCD.
    *   this warrants a change in this function,
    *   as CS1 and CS2 are now active low.  Do
    *   not change functionality anywhere else.
    *
    **************************************/
    if (value) switch (cmd) {
        case E:
            sysreg_bit_set (sysreg_FLAGS, FLG5);
            //SRU(HIGH, DAI_PB01_I);
```

```
            break;
        case CS1:
            sysreg_bit_clr (sysreg_FLAGS, FLG2);
            //SRU(LOW, DAI_PB02_I);
            break;
        case CS2:
            sysreg_bit_clr (sysreg_FLAGS, FLG3);
            //SRU(LOW, DAI_PB03_I);
            break;
        case DI:
            sysreg_bit_set (sysreg_FLAGS, FLG4);
            //SRU(HIGH, DAI_PB05_I);
            break;
        default:
            break;
    }
    else switch (cmd) {
        case E:
            sysreg_bit_clr (sysreg_FLAGS, FLG5);
            //SRU(LOW, DAI_PB01_I);
            break;
        case CS1:
            sysreg_bit_set (sysreg_FLAGS, FLG2);
            //SRU(HIGH, DAI_PB02_I);
            break;
        case CS2:
            sysreg_bit_set (sysreg_FLAGS, FLG3);
            //SRU(HIGH, DAI_PB03_I);
            break;
        case DI:
            sysreg_bit_clr (sysreg_FLAGS, FLG4);
            //SRU(LOW, DAI_PB05_I);
            break;
        default:
            break;
    }
}

/* Name: lcd_init
 * Author: Isaac Jones & Greg McCoy
 * Called By: main()
 * Function: initializes the LCD module.  Sets the
 *      SPI baud rate to the maximum usable by the
 *      LCD module.  Also initializes the E signal
 *      so that lcd_send_cmd will function correctly.
 *      Turns both LCD columns on, and initializes
 *      the values in the X, Y, and Z registers to
 *      zero so that the other lcd functions will
 *      work as expected.  NOTE: May want to add
 *      calls to lcd_clear and lcd_update so that
 *      the LCD is garaunteed to boot up in a clean
 *      state.
 * Usage: lcd_init()
 * Changes to state: Drastically alters the state of
 *      the LCD Module.  Also alters the state of the
 *      SPI transfer buffer, SPI control registers,
 *      and the E, CS1, CS2, RW, DI, and SO outputs.
*/
```

```
void
lcd_init(void)
{
    /* Theory:  DSP is running at 200 MHz.  This means that each
        cycle is 5 ns.  Also assuming that a nop takes exactly one
        cycle to complete, the number of cycles to delay can be
        computer as TIME_TO_DELAY/5
        */
    int i;
    // Initialize all of the various pins as output
    /* Need to change these initializations to work with the production board. */

    sysreg_bit_set (sysreg_FLAGS, FLG5O); // AD13, used as E (FLAG5)
    //SRU(HIGH, PBEN01_I); // DAI Pin 1, used as E
    sysreg_bit_set (sysreg_FLAGS, FLG2O); // AD10, used as CS1 (FLAG2)
    //SRU(HIGH, PBEN02_I); // DAI Pin 2, used as CS1
    sysreg_bit_set (sysreg_FLAGS, FLG3O); // AD11, used as CS2 (FLAG3)
    //SRU(HIGH, PBEN03_I); // DAI Pin 3, used as CS2
    sysreg_bit_set (sysreg_FLAGS, FLG4O); // AD12, used as D/nI (FLAG4)
    //SRU(HIGH, PBEN05_I); // DAI Pin 5, used as RS and D/nI

    lcd_ctrl(SR, 0);
    lcd_ctrl(E, 0); // set E low at the initialization

    /* Begin LCD initialization */

    *pSPIFLG = DS1EN|SPIFLG1;
    *pSPIBAUD = LCD_BAUD;
    *pSPICTL = DMISO|SPIMS|SPIEN|TIMOD1|LSBF;

    lcd_send_cmd(SET_POWER, BOTH_COLUMNS, 1);
    lcd_send_cmd(SET_X, BOTH_COLUMNS, 0x00);
    lcd_send_cmd(SET_Y, BOTH_COLUMNS, 0x00);
    lcd_send_cmd(SET_Z, BOTH_COLUMNS, 0x00);

    /* Clear LCD Screen to eliminate random start-up data */
    lcd_clear();
    lcd_update();

    //  End LCD Initialization

}

/* Name: lcd_clear
 * Author: Isaac Jones & Greg McCoy
 * Called By: lcd_init
 * Function: clears the buffer associated with the LCD
 *      display module.  lcd_update must be called
 *      immediately afterward to push changes to the LCD.
 *      Accepts no input.  Also resets the LCD index global
 *      variables to start at the beginning of the screen.
 * Usage: lcd_clear
 * Changes to state:  Clears the indexing
 *      global variables and the character array representing
 *      the current output of the LCD.  DOES NOT CHANGE THE STATE
 *      OF THE LCD ITSELF.
*/
```

```
void
lcd_clear(void)
{
    int i,j;

    lcd_cursor_x = 0;
    lcd_cursor_y = 0;
    for (i = 0; i < NUMBER_OF_MEMORY_PAGES; i++) {
        for (j = 0; j < BYTES_PER_PAGE/4; j++) {
            lcd_buffer[i][j]=0;
            lcd_buffer[i][j+BYTES_PER_PAGE/4]=0;
        }
    }
}

/* Name: lcd_print_playback
 * Author: Isaac Jones
 * Called By: Open API function.  Probably called
 *      by main program loop.
 * Function: Updates the LCD data buffer with the current
 *      playback information.  This initially clears the
 *      LCD, then adds the relevant data to ensure that
 *      the correct data is displayed on the LCD.  lcd_printf
 *      is used to print the metadata to the screen, and
 *      function-internal algorithms print the remaining
 *      information.  Metadata displayed is Song Name,
 *      Artist, and Album (EG. Beethoven's 9th, Ludwig
 *      von Beethoven, Beethoven's Symphonies).  As of 04/21,
 *      is converted to work with the new memory management
 *      scheme.
 * Usage: lcd_print_playback(
 *      data (character pointer, points to an array of 3
 *          arrays of 21 characters),
 *      progress (integer from 1 to 100 used to represent
 *          percentage of playback complete));
 * Changes to state:  Changes the state of the LCD buffer
 *      but does NOT change the actual LCD.  Changes the
 *      state of the indexing global variale lcd_buffer_y
 *      and lcd_buffer_x.
*/

void
lcd_print_playback(int progress)
{
    int i,j = 0;

    audio_get_samples (display_samples, 128);

    //print the sound data
    for (i = 0; i < 128; i++) {
        if(display_samples[i] < -7) {
            lcd_buffer[2][i/4] |= (0xFF << (3-i%4)*8);
            lcd_buffer[3][i/4] |= ((0xFF >> (7+display_samples[i]%8)) << (3-i%4)*8);
        } else if(display_samples[i] >= -7 && display_samples[i] < 0) {
            lcd_buffer[2][i/4] |= ((0xFF >> (7+display_samples[i])) << (3-i%4)*8);
        } else if(display_samples[i] <= 7 && display_samples[i] > 0) {
```

```
            lcd_buffer[1][i/4] |= (((0xFF << (8-display_samples[i]))&0xFF)  << (3-
i%4)*8);
        } else if(display_samples[i] > 7) {
            lcd_buffer[0][i/4] |= (((0xFF << (8-display_samples[i]%8))&0xFF) << (3-
i%4)*8);
            lcd_buffer[1][i/4] |= (0xFF << (3-i%4)*8);
        }
        lcd_buffer[1][i/4] |= (0x80 << (3-i%4)*8);
        lcd_buffer[2][i/4] |= (0x01 << (3-i%4)*8);
    }

    //print the progress data
    lcd_buffer[4][3] |= (0x7e << 16);
    lcd_buffer[4][28] |= (0x7e << 24);

    lcd_buffer[4][3] |= (progress > 0) ? (0x7e << 8) : (0x42 << 8);
    lcd_buffer[4][3] |= (progress > 1) ? (0x7e) : (0x42);

    j = 4;
    for (i = 3; i <= 97; i+=4) {
        if (i < progress) {
            lcd_buffer[4][j] |= (0x7e << 24);
        } else {
            lcd_buffer[4][j] |= (0x42 << 24);
        }
        if (i+1 < progress) {
            lcd_buffer[4][j] |= (0x7e << 16);
        } else {
            lcd_buffer[4][j] |= (0x42 << 16);
        }
        if (i+2 < progress) {
            lcd_buffer[4][j] |= (0x7e << 8);
        } else {
            lcd_buffer[4][j] |= (0x42 << 8);
        }
        if (i+3 < progress) {
            lcd_buffer[4][j] |= 0x7e;
        } else {
            lcd_buffer[4][j] |= 0x42;
        }
        j++;
    }

    lcd_cursor_y = 5;
    lcd_printf(flac_get_title(), 0, 0);
    lcd_cursor_y++;
    lcd_cursor_x = 0;
    lcd_printf(flac_get_album(), 0, 0);
    lcd_cursor_y++;
    lcd_cursor_x = 0;
    lcd_printf(flac_get_artist(), 0, 0);
    lcd_cursor_y++;
    lcd_cursor_x = 0;
}

/* Name: lcd_print_directories
 * Author: Isaac Jones, Brett Mravec
 * Called By: Unknown, open API function
```

```
 * Function: Print the directory browsing user
 *      interface to the LCD screen.  Also
 *      highlights the current "active row."
 *      Recieves an array of filenames and an
 *      integer representing the active line.  Like
 *      lcd_print_playback, this function clears the
 *      lcd buffer before it begin printing to elminate
 *      artifacting and lcd_update must be explicitly
 *      called before data will appear on the display.
 * Usage: lcd_print_directories(
 *      files (character pointer that points to an array
 *          of 8 arrays of 21 characters representing the
 *          files to be displayed.),
 *      active_line (integer representing the active line
 *          that will appear highlighted on the display));
 * Changes to state:  Changes the state of the lcd_buffer
 *      array and the two lcd_buffer indexes.
*/

void
lcd_print_directories(FileEntry *files, char active_line)
{
    int i = 0;
    int j = 0;

    lcd_clear();

    if (active_line > 7) {
        active_line = 7;
    }

    if (active_line < 0) {
        active_line = 0;
    }

    for (i = 0; i < 8; i++) {
        lcd_printf (files[i].name, i == active_line, 0);
        lcd_printf ("\n", 0, 0);
    }
}

/* Name: lcd_update
 * Author: Isaac Jones
 * Called By: lcd_init and lcd interrupt handler
 * Function: Pushes the data contained within the lcd buffer
 *      to the display itself.  Does not change the data in
 *      the lcd buffer in any way.  Changed as of 4/21 to use
 *      new low-memory impact lcd_buffer
 * Usage: lcd_update
 * Changes to state:  Changes the state of the lcd screen and
 *      the lcd control and data signals.  No internal memory
 *      is changed.
*/

void
lcd_update(void)
{
    *pSPIFLG = DS1EN|SPIFLG1;
```

```
    *pSPIBAUD = LCD_BAUD;
    *pSPICTL = DMISO|SPIMS|SPIEN|TIMOD1|LSBF;

    lcd_send_cmd(SET_Z, BOTH_COLUMNS, 0);

    int i,j;
    for (i=0;i<NUMBER_OF_MEMORY_PAGES;i++) {
        lcd_send_cmd(SET_X, BOTH_COLUMNS,(char) i);
        delay_processor(BRIEF_DELAY);
        lcd_send_cmd(SET_Y, BOTH_COLUMNS,0);
        delay_processor(BRIEF_DELAY);
        for (j=0;j<BYTES_PER_PAGE/4;j++) {
            lcd_send_cmd(WRITE_DATA, CS1, (char)((lcd_buffer[i][j]>>24) & 0xFF));
            delay_processor(BRIEF_DELAY);
            lcd_send_cmd(WRITE_DATA, CS1, (char)((lcd_buffer[i][j]>>16) & 0xFF));
            delay_processor(BRIEF_DELAY);
            lcd_send_cmd(WRITE_DATA, CS1, (char)((lcd_buffer[i][j]>>8) & 0xFF));
            delay_processor(BRIEF_DELAY);
            lcd_send_cmd(WRITE_DATA, CS1, (char)(lcd_buffer[i][j] & 0xFF));
            delay_processor(BRIEF_DELAY);
            lcd_send_cmd(WRITE_DATA, CS2,
(char)((lcd_buffer[i][j+BYTES_PER_PAGE/4]>>24) & 0xFF));
            delay_processor(BRIEF_DELAY);
            lcd_send_cmd(WRITE_DATA, CS2,
(char)((lcd_buffer[i][j+BYTES_PER_PAGE/4]>>16) & 0xFF));
            delay_processor(BRIEF_DELAY);
            lcd_send_cmd(WRITE_DATA, CS2,
(char)((lcd_buffer[i][j+BYTES_PER_PAGE/4]>>8) & 0xFF));
            delay_processor(BRIEF_DELAY);
            lcd_send_cmd(WRITE_DATA, CS2, (char)(lcd_buffer[i][j+BYTES_PER_PAGE/4] &
0xFF));
            delay_processor(BRIEF_DELAY);
        }
    }
}

/* Name: lcd_printf
 * Author: Isaac Jones & Greg McCoy & Brett Mravec
 * Called By: lcd_print_directories lcd_print_playback
 * Function: Emulates the stdio function printf by printing
 *      standard ASCII characters to the lcd buffer.  Accepts
 *      the standard, not extended, ACSII and 11 additional
 *      characters appended to the end of the table.
 * Usage: lcd_printf(
 *      string (character pointer to a null-terminated string
 *          of no more than 21 characters));
 * Changes to state:  Changes the state of the lcd buffer and
 *      the two buffer index variables.  Does not change any
 *      externally visible states.
*/

void
lcd_printf(char string[], char inverted, char wrap)
{
    int byte;
    int index = 0;
//    while (string[index] != 0) {
    for (index = 0; string[index] != 0; index++) {
```

```
        // line feed
        if (string[index] == '\n') {
            lcd_cursor_x = 0;
            lcd_cursor_y++;
        } else if (lcd_cursor_x * 4 >= LCD_WIDTH - CHAR_WIDTH - 1) {
            if (wrap) {
                lcd_cursor_x = 0;
                lcd_cursor_y++;
                index--;
            }
        } else {
            if (lcd_cursor_x % 3 == 0) {
                if (inverted) {
                    lcd_buffer[lcd_cursor_y][lcd_cursor_x] = ~font[string[index]][0];
                    lcd_buffer[lcd_cursor_y][lcd_cursor_x+1] =
~font[string[index]][1];
                } else {
                    lcd_buffer[lcd_cursor_y][lcd_cursor_x] = font[string[index]][0];
                    lcd_buffer[lcd_cursor_y][lcd_cursor_x+1] = font[string[index]][1];
                }
                lcd_cursor_x++;
            } else {
                if (inverted) {
                    lcd_buffer[lcd_cursor_y][lcd_cursor_x] = (~font[string[index]][0]
& 0xffff0000) >> 16 |
                        (0xffff0000 & lcd_buffer[lcd_cursor_y][lcd_cursor_x]);
                    lcd_buffer[lcd_cursor_y][lcd_cursor_x+1] =
(~font[string[index]][0] & 0xffff) << 16;
                    lcd_buffer[lcd_cursor_y][lcd_cursor_x+1] |=
(~font[string[index]][1] & 0xffff0000) >> 16;
                } else {
                    lcd_buffer[lcd_cursor_y][lcd_cursor_x] |= (font[string[index]][0]
& 0xffff0000) >> 16;
                    lcd_buffer[lcd_cursor_y][lcd_cursor_x+1] = (font[string[index]][0]
& 0xffff) << 16;
                    lcd_buffer[lcd_cursor_y][lcd_cursor_x+1] |=
(font[string[index]][1] & 0xffff0000) >> 16;
                }
                lcd_cursor_x += 2;
            }
        }

        //index++;

        if (lcd_cursor_y >= LCD_HEIGHT / CHAR_HEIGHT) {
            lcd_cursor_y = 0;
            lcd_cursor_x = 0;
            break;
        }
    }
}

/*  This is the font table used to store the ASCII to
    LCD Font Mapping.  Additional Characters should be
    added to the end of the table. */

const unsigned int font[][2] = {
// ASCII symbols
```

```
{0x00000000,0x00000000},   //   0x 0 0
{0x00641804,0x64180000},   //   0x 1 1
{0x003c4040,0x207c0000},   //   0x 2 2
{0x000c3040,0x300c0000},   //   0x 3 3
{0x003c4030,0x403c0000},   //   0x 4 4
{0x00003e1c,0x08000000},   //   0x 5 5
{0x00041e1f,0x1e040000},   //   0x 6 6
{0x00103c7c,0x3c100000},   //   0x 7 7
{0x0020403e,0x01020000},   //   0x 8 8
{0x00221408,0x14220000},   //   0x 9 9
{0x00003828,0x38000000},   //   0x a 10
{0x00001038,0x10000000},   //   0x b 11
{0x00000010,0x00000000},   //   0x c 12
{0x00087808,0x00000000},   //   0x d 13
{0x00001515,0x0a000000},   //   0x e 14
{0x007f7f09,0x09010000},   //   0x f 15
{0x0010207f,0x01010000},   //   0x10 16
{0x00040400,0x011f0000},   //   0x11 17
{0x00001915,0x12000000},   //   0x12 18
{0x00406050,0x48440000},   //   0x13 19
{0x00060909,0x06000000},   //   0x14 20
{0x000f0201,0x01000000},   //   0x15 21
{0x0000011f,0x01000000},   //   0x16 22
{0x0044444a,0x4a510000},   //   0x17 23
{0x0014741c,0x17140000},   //   0x18 24
{0x00514a4a,0x44440000},   //   0x19 25
{0x00000004,0x04040000},   //   0x1a 26
{0x00007c54,0x54440000},   //   0x1b 27
{0x0008082a,0x1c080000},   //   0x1c 28
{0x007c007c,0x447c0000},   //   0x1d 29
{0x0004027f,0x02040000},   //   0x1e 30
{0x0010207f,0x20100000},   //   0x1f 31
{0x00000000,0x00000000},   //   0x20 32
{0x0000006f,0x00000000},   // ! 0x21 33
{0x00000700,0x07000000},   // " 0x22 34
{0x00147f14,0x7f140000},   // # 0x23 35
{0x00000704,0x1e000000},   // $ 0x24 36
{0x00231308,0x64620000},   // % 0x25 37
{0x00364956,0x20500000},   // & 0x26 38
{0x00000007,0x00000000},   // ' 0x27 39
{0x00001c22,0x41000000},   // ( 0x28 40
{0x00004122,0x1c000000},   // ) 0x29 41
{0x0014083e,0x08140000},   // * 0x2a 42
{0x0008083e,0x08080000},   // + 0x2b 43
{0x00005030,0x00000000},   // , 0x2c 44
{0x00080808,0x08080000},   // - 0x2d 45
{0x00006060,0x00000000},   // . 0x2e 46
{0x00201008,0x04020000},   // / 0x2f 47
{0x003e5149,0x453e0000},   // 0 0x30 48
{0x0000427f,0x40000000},   // 1 0x31 49
{0x00426151,0x49460000},   // 2 0x32 50
{0x00214145,0x4b310000},   // 3 0x33 51
{0x00181412,0x7f100000},   // 4 0x34 52
{0x00274545,0x45390000},   // 5 0x35 53
{0x003c4a49,0x49300000},   // 6 0x36 54
{0x00017109,0x05030000},   // 7 0x37 55
{0x00364949,0x49360000},   // 8 0x38 56
{0x00064949,0x291e0000},   // 9 0x39 57
```

```
{0x00003636,0x00000000},   // : 0x3a 58
{0x00005636,0x00000000},   // ; 0x3b 59
{0x00081422,0x41000000},   // < 0x3c 60
{0x00141414,0x14140000},   // = 0x3d 61
{0x00004122,0x14080000},   // > 0x3e 62
{0x00020151,0x09060000},   // ? 0x3f 63
{0x003e415d,0x494e0000},   // @ 0x40 64
{0x007e0909,0x097e0000},   // A 0x41 65
{0x007f4949,0x49360000},   // B 0x42 66
{0x003e4141,0x41220000},   // C 0x43 67
{0x007f4141,0x413e0000},   // D 0x44 68
{0x007f4949,0x49410000},   // E 0x45 69
{0x007f0909,0x09010000},   // F 0x46 70
{0x003e4149,0x497a0000},   // G 0x47 71
{0x007f0808,0x087f0000},   // H 0x48 72
{0x0000417f,0x41000000},   // I 0x49 73
{0x00204041,0x3f010000},   // J 0x4a 74
{0x007f0814,0x22410000},   // K 0x4b 75
{0x007f4040,0x40400000},   // L 0x4c 76
{0x007f020c,0x027f0000},   // M 0x4d 77
{0x007f0408,0x107f0000},   // N 0x4e 78
{0x003e4141,0x413e0000},   // O 0x4f 79
{0x007f0909,0x09060000},   // P 0x50 80
{0x003e4151,0x215e0000},   // Q 0x51 81
{0x007f0919,0x29460000},   // R 0x52 82
{0x00464949,0x49310000},   // S 0x53 83
{0x0001017f,0x01010000},   // T 0x54 84
{0x003f4040,0x403f0000},   // U 0x55 85
{0x000f3040,0x300f0000},   // V 0x56 86
{0x003f4030,0x403f0000},   // W 0x57 87
{0x00631408,0x14630000},   // X 0x58 88
{0x00070870,0x08070000},   // Y 0x59 89
{0x00615149,0x45430000},   // Z 0x5a 90
{0x003c4a49,0x291e0000},   // [ 0x5b 91
{0x00020408,0x10200000},   // \ 0x5c 92
{0x0000417f,0x00000000},   // ] 0x5d 93
{0x00040201,0x02040000},   // ^ 0x5e 94
{0x00404040,0x40400000},   // _ 0x5f 95
{0x00000003,0x04000000},   // ` 0x60 96
{0x00205454,0x54780000},   // a 0x61 97
{0x007f4844,0x44380000},   // b 0x62 98
{0x00384444,0x44200000},   // c 0x63 99
{0x00384444,0x487f0000},   // d 0x64 100
{0x00385454,0x54180000},   // e 0x65 101
{0x00087e09,0x01020000},   // f 0x66 102
{0x000c5252,0x523e0000},   // g 0x67 103
{0x007f0804,0x04780000},   // h 0x68 104
{0x0000447d,0x40000000},   // i 0x69 105
{0x00204044,0x3d000000},   // j 0x6a 106
{0x00007f10,0x28440000},   // k 0x6b 107
{0x0000417f,0x40000000},   // l 0x6c 108
{0x007c0418,0x04780000},   // m 0x6d 109
{0x007c0804,0x04780000},   // n 0x6e 110
{0x00384444,0x44380000},   // o 0x6f 111
{0x007c1414,0x14080000},   // p 0x70 112
{0x00081414,0x187c0000},   // q 0x71 113
{0x007c0804,0x04080000},   // r 0x72 114
{0x00485454,0x54200000},   // s 0x73 115
```

```
{0x00043f44,0x40200000},   // t 0x74 116
{0x003c4040,0x207c0000},   // u 0x75 117
{0x001c2040,0x201c0000},   // v 0x76 118
{0x003c4030,0x403c0000},   // w 0x77 119
{0x00442810,0x28440000},   // x 0x78 120
{0x000c5050,0x503c0000},   // y 0x79 121
{0x00446454,0x4c440000},   // z 0x7a 122
{0x00000836,0x41410000},   // { 0x7b 123
{0x0000007f,0x00000000},   // | 0x7c 124
{0x00414136,0x08000000},   // } 0x7d 125
{0x00040204,0x08040000},   // ~ 0x7e 126
{0x007f7f7e,0x7e7e0000},   //   0x7f 127
{0x00607e06,0x667e0000},   //   0x80 128  musical note
{0x007F7f7e,0x7e7e0000},   //   0x81 129  folder
{0x007E4242,0x467C0000},   //   0x82 130  generic file
{0x00000A44,0x2A200000},   //   0x83 131  dead face left
{0x202A440A,0x00000000},   //   0x84 132  dead face right
{0x0000042E,0x44400000},   //   0x85 133  happy face left
{0x40442E04,0x00000000},   //   0x86 134  happy face right
{0x0004027F,0x02040000},   //   0x87 135  up arrow
{0x0010207F,0x20100000},   //   0x88 136  down arrow
{0x0008082A,0x1C080000},   //   0x89 137  right arrow
{0x00081c2a,0x08080000}    //   0x8a 138  left arrow
};
/*  Title:     Raw Audio Loader
 *  Version:   0.5
 *  Filename:  raw.c
 *  Authors:   Brett Mravec
 *  Purpose:   load raw data from file to buffer
 *  Date:      23 April 2009
 */

#include "raw.h"
#include "audio.h"
#include "fat.h"
#include "bit-reader.h"

static BitReader br;

static short a_sound[100] = { 0, 4114, 8213, 12280, 16297, 20251, 24125, 27903, 31571,
35115, 38520, 41773, 44861, 47772, 50495, 53018, 55333, 57428, 59297, 60932, 62327,
63476, 64374, 65018, 65405, 65535, 65405, 65018, 64374, 63476, 62327, 60932, 59297,
57428, 55333, 53018, 50495, 47772, 44861, 41773, 38520, 35115, 31571, 27903, 24125,
20251, 16297, 12280, 8213, 4114, 0, -4114, -8213, -12280, -16297, -20251, -24125, -
27903, -31571, -35115, -38520, -41773, -44861, -47772, -50495, -53018, -55333, -57428,
-59297, -60932, -62327, -63476, -64374, -65018, -65405, -65535, -65405, -65018, -
64374, -63476, -62327, -60932, -59297, -57428, -55333, -53018, -50495, -47772, -44861,
-41773, -38520, -35115, -31571, -27903, -24125, -20251, -16297, -12280, -8213, -4114
};
static char a_pos;

char
raw_init (void)
{
    bit_reader_init (&br);

    audio_init (0);
    a_pos = 0;
```

```
    return 0;
}

char
raw_read_frame (void)
{
    short *frame = audio_get_buffer ();

    if (!frame) {
        return 0;
    }

    int i;
//    for (i = 0; i < 2304; i++) {
//    for (i = 0; i < 1152; i++) {
//        frame[i] = bit_reader_get (&br, 8) | (bit_reader_get_s (&br, 8) << 8);
//        frame[i] = (frame[i] / 4) + (65535 / 4);
//        frame[2*i] = frame[i];
//    }
    for (i = 0; i < 1152; i++) {
        //frame[i] = a_sound[a_pos = (a_pos + 1) % 100] / 4 + (65535 / 4);
        frame[i] = a_sound[a_pos = (a_pos + 1) % 100];
        frame[2*i] = frame[i];
//        frame[2*i] = frame[i];
    }

    audio_buffer_set_full ();

//    return fat_eof ();
    return 0;
}
/*  Title:     SD Card Interface
 *  Version:   1.0
 *  Filename:  sd.c
 *  Authors:   Brett Mravec
 *  Purpose:   Low-level SD card communication
 *  Date:      25 Mar 2009
 */

#include <Cdef21262.h>
#include <def21262.h>
#include <sysreg.h>

#include "spi.h"
#include "sd.h"

static int sd_baud;
static int crcerr_cnt;
static int cmd_crcerr_cnt;
static int tot_cnt;

char
sd_crc7 (char *data, int length)
{
    int i;
    char crc = 0, mask;
```

```
    for (i = 0; i < length; i++) {
        mask = 0x80;

        while (mask) {
            crc = (data[i] & mask) ? crc << 1 | 1 : crc << 1;
            if (crc & 0x80)
                crc ^= 0x89;
            mask >>= 1;
        }
    }

    for (i = 0; i < 7; i++) {
        crc <<= 1;
        if (crc & 0x80)
            crc ^= 0x89;
    }

    return crc;
}

short
sd_crc16 (char *data, int length)
{
    int i, crc = 0, mask;

    for (i = 0; i < length; i++) {
        mask = 0x80;
        while (mask) {
            crc = (data[i] & mask) ? crc << 1 | 1 : crc << 1;
            if (crc & 0x10000)
                crc ^= 0x11021;
            mask >>= 1;
        }
    }

    for (i = 0; i < 16; i++) {
        crc <<= 1;
        if (crc & 0x10000)
            crc ^= 0x11021;
    }

    return crc;
}

char
sd_wait_for_tt (void)
{
    int i = 0;
    char recv;

    do {
        recv = spi_xfer (0xff);
    } while (recv != 0xfe && recv != 0x08 && i++ < 5000);

    return recv;
}

int
```

```
sd_init (void)
{
    int i;
    char recv, status = 1;
    crcerr_cnt = 0;
    cmd_crcerr_cnt = 0;
    tot_cnt = 0;

    // SPI Clock Range < 400 kHz
    *pSPIBAUD = SD_INIT_BAUD;

    // SPI Configuration
    *pSPICTL = SD_SPICTL;

    sysreg_bit_set (sysreg_FLAGS, FLG15O);
    sysreg_bit_set (sysreg_FLAGS, FLG15);

    for (i = 0; i < 11; i++)
        spi_xfer (0xff);

    sysreg_bit_clr (sysreg_FLAGS, FLG15);

    for (i = 0; i < 3; i++)
      spi_xfer (0xff);

    // Send Reset to SD
    do {
        recv = sd_send_cmd (CMD0_GO_IDLE_STATE, 0);
    } while (recv != 0x01);

    while (status) {
        // Send APP
        recv = sd_send_cmd (CMD55_APP_CMD, 0);

        // Send Card Initialization
        recv = sd_send_cmd (ACMD41_SEND_OP_COND, 0);

        status = recv & 0x01;

        if (status)
            for (i = 0; i < 1000; i++)
                asm ("nop;");
    }

    i = 0;
    for (i = 0; i < 32; i++)
        recv = spi_xfer (0xff);

    sysreg_bit_set (sysreg_FLAGS, FLG15);

    return 0;
}

char
sd_read_block (int addr, char *buff)
{
    char recv, temp;
    int i;
```

```
    *(volatile int *)SPIBAUD = SD_BAUD;
    *(volatile int *)SPICTL = SD_SPICTL;

    sysreg_bit_clr (sysreg_FLAGS, FLG15);

    while (1) {
        recv = sd_send_cmd (CMD17_READ_SINGLE_BLOCK, addr);

//        if (recv & 0x60) {
//            return recv;
//        }

        if (recv != 0x01 && recv != 0x00) {
            cmd_crcerr_cnt++;
            continue;
        }

        recv = sd_wait_for_tt ();

        if (recv != 0xfe) {
            return -1;
        }

        for (i = 0; i < 512; i++) {
            buff[i] = spi_xfer (0xff);
        }

        int crc = sd_crc16 (buff, 512);
        int rcrc;

        // eat off crc-16 that we are not using
        rcrc = spi_xfer (0xff) << 8;
        rcrc |= spi_xfer (0xff);

        if (rcrc == crc)
            break;
        //break;
        crcerr_cnt++;
    }

    tot_cnt++;

    for (i = 0; i < 32; i++) {
        spi_xfer (0xff);
    }

    sysreg_bit_set (sysreg_FLAGS, FLG15);

    return recv;
}

char
sd_send_cmd (char command, int arg)
{
    // crc only matters for CMD0 and it has a fixed crc of 0x95 so
    // we hard code it and send the same one all the time
    char crcByte = 0x95;
```

```
    char recv;
    char i = 0;

    char send_buff[6];

    // first byte has "01" with 6-bit command
    send_buff[0] = 0x40 | (command & 0x3f);
    send_buff[1] = arg >> 24;
    send_buff[2] = arg >> 16;
    send_buff[3] = arg >> 8;
    send_buff[4] = arg;

    send_buff[5] = sd_crc7 (send_buff, 5) << 1 | 1;

    for (i = 0; i < 6; i++)
        recv = spi_xfer (send_buff[i]);

    do {
        recv = spi_xfer (0xff);
    } while ((recv & 0x80) && (i++ < 50));

    return recv;
}

void
sd_flush (void)
{
    int i;

    sysreg_bit_clr (sysreg_FLAGS, FLG15);

    *(volatile int *)SPIBAUD = SD_BAUD;
    *(volatile int *)SPICTL = SD_SPICTL;

    for (i = 0;i < 524; i++) {
        spi_xfer (0xff);
    }

    sysreg_bit_set (sysreg_FLAGS, FLG15);
}
/*  Title:     SPI
 *  Version:   1.0
 *  Filename:  spi.c
 *  Authors:   Brett Mravec
 *  Purpose:   SPI communication routines
 *  Date:      6 Apr 2009
 */

#include <Cdef21262.h>
#include <def21262.h>

#include "spi.h"

/*  Function Name:  char spi_xfer(char b)
 *  Author(s):      Brett Mravec
 *  Purpose:        Performs a full duplex SPI communication
 *  Parameters:     char b: byte to send
 *  Returns:        char:   byte received
```

```
 */
char
spi_xfer (char b)
{
    // Wait for write buffer to be empty
    while (*pSPISTAT & 0x08);

    *pTXSPI = b;

    // wait for shift-reg to finish shifting data
    while (!(*pSPISTAT & 1));

    return *pRXSPI;
}
/*  Title:     Global Utilities
 *  Version:   1.0
 *  Filename:  util.c
 *  Authors:   Greg McCoy, Isaac Jones, Brett Mravec
 *  Purpose:   Common utilities applicable to all software modules
 *  Date:      2 Apr 2009
 */

#include "util.h"

/*  Function Name:  void delay_processor(int count)
 *  Author(s):      Isaac Jones
 *  Purpose:        Delays processor by specified time
 *  Parameters:     count:  approximate delay time in ns
 *  Returns:        nothing
 */
void
delay_processor (int count)
{
    int i = 0;

    for (i = 0; i < count/TIME_PER_CYCLE; i++) {
        asm("nop;");
    }

    return;
}


/*  Function Name:  void memcpy(char *dest, char *src, int len)
 *  Author(s):      Brett Mravec
 *  Purpose:        Copies memory
 *  Parameters:     dest: destination pointer
 *                  src:  source pointer
 *                  len:  number of bytes to copy
 *  Returns:        nothing
 */
void
memcpy (char *dest, char *src, int len)
{
    int i;
    for (i = 0; i < len; i++) {
        dest[i] = src[i];
    }
```

```
}

/*  Function Name:  void ends_with (char *string, char *suffix)
 *  Author(s):      Brett Mravec
 *  Purpose:        Checks the suffix of a string
 *  Parameters:     string: string to check ending of
 *                  suffix: ending to check for
 *  Returns:        1 if string ends with suffix
 *                  0 else
 */
char
ends_with (char *string, char *suffix)
{
    int suf_end = 0, str_end = 0;

    while (string [str_end++]);
    while (suffix [suf_end++]);

    str_end--;
    suf_end--;

    if (str_end < suf_end) {
        return 0;
    }

    while (suffix[suf_end] != '.' && string[str_end] != '.' &&
           str_end > 0 && suf_end > 0) {
        if (suffix[suf_end] != string[str_end]) {
            return 0;
        }

        suf_end--;
        str_end--;
    }

    if (suf_end == 0 && string[str_end-1] != '.') {
        return 0;
    }

    if (suffix[suf_end] == string[str_end]) {
        return 1;
    }

    return 0;
}
```

Appendix G: FMECA Worksheet

Failure No.	Failure Mode	Possible Causes	Failure Effects	Method of Detection	Criticality	Remarks
A1	Charging Circuit Outputs 0 V	Failure of Charging IC	Battery will not charge	Observation	Low	
A2	Charging Circuit Sources large current	Failure of Charging IC	Damaged battery	Excessive Heat	High	Short from power outlet to battery
A3	5 V Boost output = 0 V	Failure of LT1302 or associated components	LCD and Audio wont work	Observation	Medium	Causes failures of other components
A4	3.3 V Regulator outputs 0 V	Failure of TPS63030 or associated components	DSP will not run	Observation	Medium	Causes system to fail
B1	Op-amp outputs 0V or unpredictable output	Failure of SSM2135 or associated circuitry	No or noise audio output	Observation	Low/High	Can either produce no audio or random audio possibly damaging hearing of user

Failure No.	Failure Mode	Possible Causes	Failure Effects	Method of Detection	Criticality	Remarks
B2	DAC outputs 0V or unpredictable output	Failure of DAC or associated circuitry	No or noise audio output	Observation	Low/High	Can either produce no audio or random audio possibly damaging hearing of user
C1	Level Translator outputs incorrent levels (< 5 V)	Failure of CD4504B	LCD displays incorrect results	Observation	Low	Only affects LCD subsystem
C2	Shift register contains incorrect value	Failure of shift register	LCD is on but displays incorrect data	Observation	Low	
C3	LCD displays erratic behavior	Failure of LCD module	Erroneous LCD output	Observation	Low	Easily replaceable part
D1	Clock outputs invalid waveform	Crystal Failure	Erratic DSP behavior	Observation	Medium	Effects the rest of the system
D2	DSP outputs incorrect value on pins	Chip failure	Erratic system behavior	Observation	Medium/High	Could cause bad audio to occur else it is at a medium criticality level